# โครงสร้างของโปรแกรมภาษา C/C++ สำหรับชุดกล่องสมองกล

IPST MicroBOX SE

Secondary Education

**วิลาศิณี วิสิทธิ์กาศ**

# โครงสร้างของโปรแกรมภาษา C/C++ สำหรับชุดกล่องสมองกล IPST MicroBOX SE

วิลาศิณี วิสิทธิ์กาศ



**ใครควรใช้หนังสือเล่มนี้**

1. นักเรียน นิสิต นักศึกษา และบุคคลทั่วไปที่มีความสนใจในการนำไมโครคอนโทรลเลอร์ไปประยุกต์ใช้ในการทดลองเกี่ยวกับการทำงานของระบบอัตโนมัติ หรือสนใจในการเรียนรู้และทดลองไมโครคอนโทรลเลอร์ในแนวทางใหม่ที่ใช้กิจกรรมหุ่นยนต์อัตโนมัติเป็นสื่อ

2. สถาบันการศึกษา โรงเรียน วิทยาลัย มหาวิทยาลัย ที่มีการเปิดการเรียนการสอนวิชาอิเล็กทรอนิกส์หรือภาควิชาวิศวกรรมอิเล็กทรอนิกส์และคอมพิวเตอร์

3. คณาจารย์ที่มีความต้องการศึกษา และเตรียมการเรียนการสอนวิชาไมโครคอนโทรลเลอร์ รวมถึงวิทยาศาสตร์ประยุกต์ที่ต้องการบูรณาการความรู้ทางอิเล็กทรอนิกส์-ไมโครคอนโทรลเลอร์-การเขียนโปรแกรมคอมพิวเตอร์-การทดลองทางวิทยาศาสตร์ ในระดับมัธยมศึกษา อาชีวศึกษา และปริญญาตรี

ดำเนินการจัดพิมพ์และจำหน่ายโดย

**บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด**

108 ซ.สุขุมวิท 101/2 ถ.สุขุมวิท แขวงบางนา เขตบางนา กรุงเทพฯ 10260

โทรศัพท์ 0-2747-7001-4

โทรสาร 0-2747-7005

# คำชี้แจงจากคณะผู้เขียน/เรียบเรียง

การนำเสนอข้อมูลเกี่ยวกับข้อมูลทางเทคนิคและเทคโนโลยีในหนังสือเล่มนี้ เกิดจากความต้องการที่จะอธิบายกระบวนการและหลักการทำงาน ของอุปกรณ์ในภาพรวมด้วยถ้อยคำที่ง่ายเพื่อสร้างความเข้าใจแก่ผู้อ่าน ดังนั้นการแปลคำศัพท์ทางเทคนิคหลายๆ คำอาจไม่ตรงตามข้อบัญญัติของราชบัณฑิตยสถาน และมีหลายๆ คำที่ยังไม่มีการบัญญัติอย่างเป็นทางการ คณะผู้เขียนจึงขออนุญาตบัญญัติศัพท์ขึ้นมาใช้ในการอธิบาย โดยมีข้อจำกัดเพื่ออ้างอิงในหนังสือเล่มนี้เท่านั้น

ทั้งนี้สาเหตุหลักของข้อชี้แจงนี้มาจาก การรวบรวมข้อมูลของอุปกรณ์ในระบบสมองกลฝังตัวและเทคโนโลยีหุ่นยนต์สำหรับการศึกษาเพื่อนำมาเรียบเรียงเป็นภาษาไทยนั้นทำได้ไม่ง่ายนัก ทางคณะผู้เขียนต้องทำการรวบรวมและทดลองเพื่อให้แน่ใจว่า ความเข้าใจในกระบวนการทำงานต่างๆ นั้นมีความคลาดเคลื่อนน้อยที่สุด

เมื่อต้องทำการเรียบเรียงออกมาเป็นภาษาไทย ศัพท์ทางเทคนิคหลายคำมีความหมายที่ทับซ้อนกันมาก การบัญญัติศัพท์จึงเกิดจากการปฏิบัติจริงร่วมกับความหมายทางภาษาศาสตร์ ดังนั้นหากมีความคลาดเคลื่อนหรือผิดพลาดเกิดขึ้น ทางคณะผู้เขียนขอน้อมรับและหากได้รับคำอธิบายหรือชี้แนะจากท่านผู้รู้จะได้ทำการชี้แจงและปรับปรุงข้อผิดพลาดที่อาจมีเหล่านั้นโดยเร็วที่สุด

ทั้งนี้เพื่อให้การพัฒนาสื่อทางวิชาการ โดยเฉพาะอย่างยิ่งกับความรู้ของเทคโนโลยีสมัยใหม่สามารถดำเนินไปได้อย่างต่อเนื่อง ภายใต้การมีส่วนร่วมของผู้รู้ในทุกภาคส่วน

*บริษัท อินโนเวตีฟ เอ็กเพอริเมนต์ จำกัด*

---

*หนังสือเล่มนี้เป็นส่วนหนึ่งของชุดกล่องสมองกล IPST-MicroBOX Seconday Edition (SE) โดยใช้ประกอบในการเรียนรู้และฝึกฝนเกี่ยวกับการเขียนและพัฒนาโปรแกรมภาษา C/C++ เพื่อนำมาพัฒนาโปรแกรมสำหรับกำหนดและควบคุมการทำงานของแผงวงจรต่างๆ ในชุดกล่องสมองกล IPST-MicroBOX (SE)*

# สารบัญ

# บทที่ 1

# โครงสร้างภาษา C/C++ ที่ใช้ในซอฟต์แวร์ Wiring

ภาษาที่ใช้ในโปรแกรม Wiring คือภาษา C/C++ และโปรแกรมจะทำงานแบบ C++ บนฮาร์ดแวร์ Wiring I/O และแผงวงจร IPST-SE (ซึ่งก็เป็น Wiring I/O แบบหนึ่งที่มีการดัดแปลงให้ใช้ในชุดกล่องสมอง IPST-MicroBOX เพื่อนำไปสร้างโครงงานและหุ่นยนต์อัตโนมัติ) จุดเด่นของซอฟต์แวร์ Wiring และการเขียนโปรแกรมด้วยภาษา C/C++ คือ ไลบรารีของ Wiring และรูปแบบของโปรแกรมจะง่ายและสามารถใช้งานได้โดยไม่จำเป็นต้องเข้าใจแนวคิดขั้นสูงเช่น คลาส (class), ออบเจ็กต์ (object) หรือพอยน์เตอร์ (pointer) แต่ยังสามารถใช้ฟังก์ชั่นเหล่านั้นได้อยู่ จึงช่วยให้โปรแกรมสั้นลงและทำความเข้าใจได้ง่ายขึ้น

## 1.1 ภาษา C/C++

ภาษาC/C++ จะเริ่มทำงานที่ฟังก์ชั่น main() ซึ่งเป็นฟังก์ชั่นหลัก เปรียบเทียบรูปแบบของโปรแกรมภาษา C และ C++ ได้ดังนี้

__ตัวอย่างโปรแกรมในภาษา C__

```
#include <stdio.h>
int main()
{
    printf("hello, world\n");
    return 0;
}
```

__ตัวอย่างโปรแกรมในภาษา C++__

```
#include <iostream>    // provides std::cout
int main()
{
    std::cout << "Hello, world!\n";
}
```

## 1.2 รูปแบบโปรแกรมภาษา C/C++ ใน Wiring

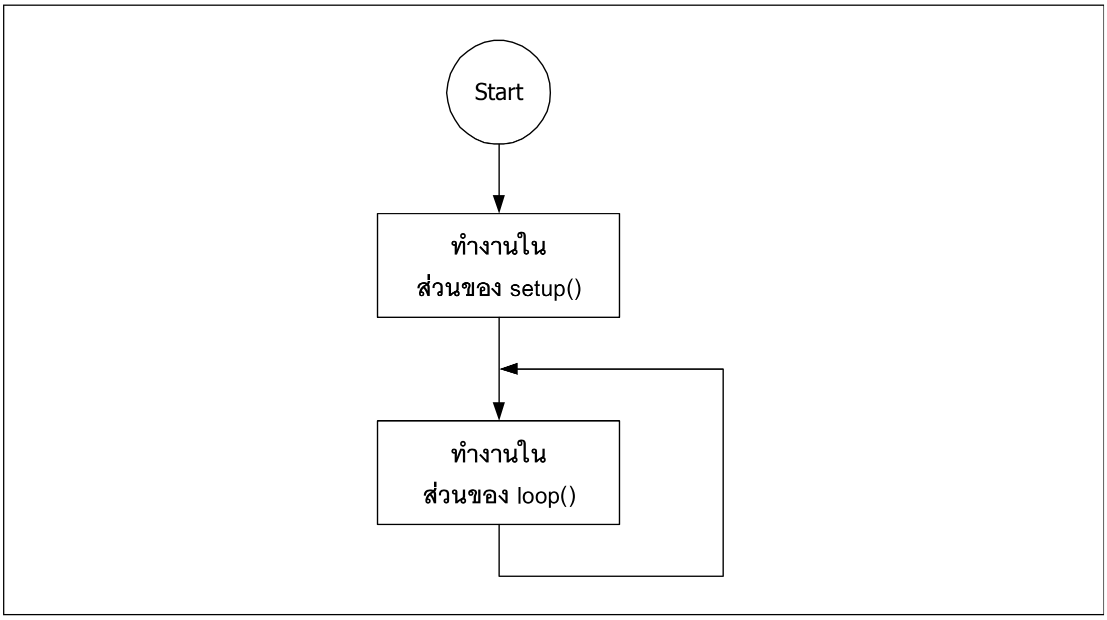
ใน Wiring มีฟังก์ชั่นหลัก 2 ฟังก์ชั่นคือ **setup()** และ **loop()** โดยไม่ต้องเขียนฟังก์ชั่น *main()* ขึ้นเองเพราะ wiring จะสร้างฟังก์ชั่น main() ให้ โดยภายในฟังก์ชั่น main() ที่สร้างขึ้นจะมีโครงสร้างดังนี้

```
int main()
{
    setup();            // เรียกฟังก์ชั่น setup() ขึ้นมาทำงาน
    for(;;)
    {
        loop();         // เรียกฟังก์ชั่น loop() ขึ้นมาทำงาน
    }
    return 0;
}
```

ใน Wiring ผู้พัฒนาโปรแกรมไม่ต้องเขียนฟังก์ชั่น main() แต่จะทำการแก้ไขเฉพาะฟังก์ชั่น setup() และ loop() ซึ่งจำเป็นต้องมี setup()และ loop() ทุกครั้งที่เขียนโปรแกรมใน wiring

### 1.2.1 ฟังก์ชั่นหลักของโปรแกรมภาษา C/C++ ใน wiring

ในรูปที่ 1-1 แสดงไดอะแกรมการทำงานของฟังก์ชั่นหลักในโปรแกรมของ Wiring




**รูปที่ 1-1 โฟลวชาร์ตแสดงการทำงานของฟังก์ชั่นหลักในโปรแกรมภาษา C/C++ ของ Wiring ซึ่งประกอบด้วย setup() และ loop()**

### 1.2.1.1 setup()

**setup()** คือฟังก์ชั่นที่จะทำงานเพียงครั้งเดียวหลังจากโปรแกรมเริ่มทำงาน จึงมักใช้ในการกำหนดค่าคงที่หรือประกาศตัวแปรหลักของโปรแกรม

### 1.2.1.2 loop()

**loop()** คือฟังก์ชั่นที่จะทำงานวนซ้ำหลายครั้ง จะเริ่มทำงานหลังจากทำงานในฟังก์ชั่นsetup() เสร็จสิ้นลง

*ทั้ง setup() และ loop() เป็นฟังก์ชั่นที่ไม่มีการคืนค่า (void) และเป็นฟังก์ชั่นที่ไม่มีพารามิเตอร์*

<u>**ตัวอย่างที่ 1-1**</u>

```
int ledPin = 17;                    // LED เชื่อมต่อกับแผงวงจร IPST-SE ที่พอร์ต 17
void setup()
{
    pinMode(ledPin, OUTPUT);        // กำหนดให้ทำงานเป็นขาพอร์ตเอาต์พุต
}
void loop()
{
    digitalWrite(ledPin, HIGH);     // ตั้งค่าให้ LED สว่าง
    delay(1000);                    // รอเพื่อให้ LED สว่างค้างไว้ประมาณ1วินาที
    digitalWrite(ledPin, LOW);      // ตั้งค่าให้ LED ดับ
    delay(1000);                    // รอเพื่อให้ LED ดับค้างไว้ประมาณ1วินาที
}
```

ตัวแปรที่ทำการประกาศให้ใช้งานเฉพาะในฟังก์ชั่น setup() จะไม่สามารถใช้งานได้ในฟังก์ชั่น loop() และเช่นเดียวกันตัวแปรที่ทำการประกาศในฟังก์ชั่น loop() จะไม่สามารถใช้งานได้ในฟังก์ชั่น setup()

หากต้องการให้ตัวแปรที่กำหนดใช้งานได้ในทั้ง 2 ฟังก์ชั่น ทำได้โดย **ประกาศตัวแปรเป็น global declaretion นั่นคือ การประกาศตัวแปรระบบ** เช่น การประกาศ int ledPin = 48 ที่บรรทัดแรก

## 1.2.2 การใช้เซมิโคลอน ;

คำสั่งของโปรแรมภาษา C/C++ จะมีสัญลักษณ์ ; หรือเซมิโคลอน (semicolon) ต่อท้ายทุกครั้ง

<u>**ตัวอย่างที่ 1-2**</u>

```
int y;
```

### 1.2.3 การใช้วงเล็บปีกกา {} (เคอร์ลีเบรซเซส : curly braces)

{} หรือวงเล็บปีกกาใช้เพื่อบอกจุดเริ่มต้นและจุดสิ้นสุดของฟังก์ชั่นและสเตตเมนต์ ซึ่งอาจมีได้หลายบรรทัด เช่น โครงสร้างของ for() และ () หรืออาจใช้ในการให้ค่าเริ่มต้นแก่ตัวแปรอะเรย์ (array)

**<u>ตัวอย่างที่ 1-3</u>**

```
int[] a = { 5, 20, 25, 45, 70 };
```

### 1.2.4 การเขียนคอมเมนต์ (comment) หรือคำอธิบายโปรแกรม

คอมเมนต์คือข้อความที่ไม่เกี่ยวข้องกับโค้ด อาจเป็นข้อความที่ผู้พัฒนาโปรแกรมเขียนเพื่อความเข้าใจของตัวเอง หรือใช้ในการอธิบายการทำงานของโปรแกรมในแต่ละบรรทัดหรือแต่ละย่อหน้า คอมไพเลอร์จะไม่แปลหรือตีความใดๆ ในส่วนของคอมเมนต์

การเขียนคอมเมนต์ในโปรแกรมภาษา C/C++ มี 2 แบบ คือ

*// comment* *เป็นการเขียนคอมเมนต์หนึ่งบรรทัด*

*/* comment */* *เป็นการเขียนคอมเมนต์หลายบรรทัด*

### 1.2.5 ไดเร็กตีฟ #include

ใช้ผนวกไฟล์ไลบรารีเข้ากับโปรแกรมในไฟล์สเก็ตช์ ไดเร็กตีฟ #include นี้มักถูกเขียนไว้บรรทัดบนสุดของโปรแกรมเสมอ

**รูปแบบ** :

```
#include <ชื่อlibaryที่ต้องการinclude>
```

**<u>ตัวอย่างที่ 1-4</u>**

```
#include <glcd.h>  //เพื่อให้สามารถใช้คำสั่งเกี่ยวกับโมดูลกราฟิก LCD สีได้
```

**หมายเหตุ** #include เป็นไดเร็กตีฟ จึงไม่ต้องลงท้ายด้วยเซมิโคลอนหรือ ;

## 1.3 ข้อมูลและตัวแปร

### 1.3.1 การประกาศตัวแปร

ในการเรียกใช้งานข้อมูลจำเป็นต้องมีตัวแปรมาเก็บข้อมูลนั้น ซึ่งต้องมีการประกาศตัวแปรก่อนจึงใช้งานได้ โดยปกติมักประกาศตัวแปรไว้ด้านบนสุดของฟังก์ชั่นหรือตอนต้นของโปรแกรม ตัวแปรที่ถูกประกาศในฟังก์ชั่นหนึ่งจะไม่สามารถเรียกใช้ได้ในอีกฟังก์ชั่นหนึ่ง ตัวแปรชนิดเดียวกันสามารถประกาศพร้อมกันหลายๆ ตัวได้ โดยใช้เครื่องหมาย , เป็นตัวแยก

สรุปชนิดของตัวแปรที่ใช้ในโปรแกรมภาษา C/C++ ใน Wiring ดังนี้

| **ชนิดของตัวแปร** | **คำอธิบาย** |
|---|---|
| int | เก็บตัวแปรชนิดตัวเลขที่ไม่มีจุดทศนิยม |
| unsigned int | เก็บตัวแปรชนิดตัวเลขที่ไม่มีจุดทศนิยม |
| long | เก็บตัวแปรชนิดตัวเลขที่ไม่มีจุดทศนิยม |
| unsigned long | เก็บตัวแปรชนิดตัวเลขที่ไม่มีจุดทศนิยม |
| char | เก็บตัวแปรชนิดตัวอักษร |
| unsigned char | เก็บตัวแปรชนิดตัวอักษร |
| float | เก็บตัวแปรชนิดตัวเลขที่มีจุดทศนิยม |
| double | เก็บตัวแปรชนิดตัวเลขที่มีจุดทศนิยม |
| boolean | เก็บตัวแปรชนิด boolean หรือ จริง (true) และ เท็จ (false) |
| byte | เก็บตัวแปรชนิด byte |

**รูปแบบ :**

datatype var;

**โดยที่** datatype คือ ชนิดของตัวแปรจะขึ้นอยู่กับชนิดของข้อมูลที่ตัวแปรนั้นเก็บค่า เช่น int, float, char

var คือ ชื่อของตัวแปร ซึ่งมีกฎในการตั้งชื่อตัวแปรดังนี้

(1) ชื่อตัวแปรสามารถใช้ตัวอักษรหรือตัวเลขหรือเครื่องหมายสัญประกาศ (เครื่องหมายขีดเส้นล่าง_ หรือ under scroll) ในการตั้งชื่อได้แต่ต้องขึ้นต้นด้วยตัวอักษรภาษาอังกฤษ หรือเครื่องหมายสัญประกาศเท่านั้น

(2) ชื่อตัวแปรไม่สามารถเว้นวรรคได้และห้ามมีเครื่องหมายต่าง ๆ ยกเว้นเครื่องหมายสัญประกาศในชื่อตัวแปร เช่น $, %, *, @, +, - ,/, ^ เป็นต้น

(3) ตัวอักษรพิมพ์ใหญ่และพิมพ์เล็กมีความแตกต่างกัน (case-sensitive)

(4) ชื่อตัวแปรห้ามเป็นคำสงวน เช่น begin, end, if เป็นต้น

**<u>ตัวอย่างที่ 1-5</u>**

```
int x;          // ประกาศตัวแปร x เป็นตัวแปรชนิดตัวเลขจำนวนเต็ม
char a,b,c;     // ประกาศตัวแปร a, b และ c  เป็นตัวแปรชนิดตัวอักษร
```

## 1.3.2 การกำหนดค่าให้ตัวแปร

การกำหนดค่าให้ตัวแปรทำได้โดยใช้เครื่องหมาย = (assignment operator) ซึ่งจะกำหนดค่าให้ตัวแปรได้ก็ต่อเมื่อชนิดข้อมูลและชนิดของตัวแปรนั้นเป็นชนิดเดียวกัน เช่น ถ้าข้อมูลเป็น interger ตัวแปรก็ควรจะเป็นชนิด interger ด้วย การกำหนดค่าให้ตัวแปรอาจทำพร้อมกับการประกาศตัวแปรก็ได้

**รูปแบบ** :

var = value;

โดยที่ value คือ ข้อมูลที่จะให้ตัวแปรนั้นเก็บค่า

var คือ ชื่อของตัวแปรที่ต้องการจะให้ค่า

**ตัวอย่างที่ 1-6**

```
int x = 10;     // ประกาศตัวแปร x เป็นตัวแปรชนิดตัวเลขจำนวนเต็ม มีค่าเริ่มต้นเท่ากับ 10
x = 20;         // กำหนดให้ค่าตัวแปร x เท่ากับ 20
```

**หมายเหตุ** : ในการประกาศตัวแปรควรกำหนดค่าตัวแปรด้วยทุกครั้งเพื่อป้องกันความผิดพลาดของโปรแกรม

## 1.3.3 ชนิดของข้อมูล

ในภาษาC/C++มีชนิดของข้อมูลอยู่หลายแบบดังต่อไปนี้

### 1.3.3.1 int

คือข้อมูลที่เป็นตัวเลขที่ไม่มีจุดทศนิยม ขนาดข้อมูลที่ตัวแปรชนิด int เก็บได้คือ 32,767 ถึง -32,768 โดยจะเก็บเป็นข้อมูลขนาด16 บิต

**ตัวอย่างที่ 1-7**

```
int x = 23;     // ประกาศตัวแปร x เป็นตัวแปรชนิดตัวเลขจำนวนเต็ม มีค่าเริ่มต้นเท่ากับ 23
```

### 1.3.3.2 unsigned int

คือข้อมูลที่เป็นตัวเลขที่ไม่มีจุดทศนิยมและไม่คิดเครื่องหมาย ข้อมูลที่ตัวแปรชนิดนี้เก็บได้คือ 0 ถึง 65,535 โดยเก็บเป็นข้อมูลขนาด16 บิต

**ตัวอย่างที่ 1-8**

```
unsigned int y = 1024;
// ประกาศตัวแปร y เป็นตัวแปรชนิดตัวเลขจำนวนเต็มไม่คิดเครื่องหมาย มีค่าเริ่มต้นเท่ากับ 1024
```

### 1.3.3.3 long

คือชนิดข้อมูลที่เป็นตัวเลขจำนวนเต็มขนาด 32 บิตแบบคิดเครื่องหมาย ข้อมูลที่เก็บได้คือ -2,147,483,648 ถึง 2,147,483,647

**ตัวอย่างที่ 1-9**

```
long x = 10;          // ประกาศตัวแปร x เป็นตัวแปรชนิด long แบบคิดเครื่องหมาย มีค่าเริ่มต้นเป็น 10
```

### 1.3.3.4 unsigned long

คือชนิดข้อมูลที่เป็นตัวเลขจำนวนเต็ม 32 บิตแบบไม่คิดเครื่องหมาย มีค่า 0 ถึง 4,294,967,295

**ตัวอย่างที่ 1-10**

```
unsigned long y = 1024;
// ประกาศตัวแปร y เป็นตัวแปรชนิด long แบบไม่คิดเครื่องหมาย มีค่าเริ่มต้นเท่ากับ 1024
```

### 1.3.3.5 float

คือชนิดข้อมูลที่เป็นตัวเลขทศนิยมขนาด 32 บิต แบบคิดเครื่องหมาย เลขทศนิยมมักใช้ในการประมาณค่าของอะนาลอกและค่าที่ต่อเนื่อง เพราะให้ค่าที่มีความละเอียดและแม่นยำมากกว่าข้อมูลแบบจำนวนเต็ม int ข้อมูลที่เก็บได้คือ $-3.4028235 \times 10^{+38}$ ถึง $+3.4028235 \times 10^{+38}$

**ตัวอย่างที่ 1-11**

```
float x = 2.5;        // ประกาศตัวแปร x เป็นตัวแปรชนิดเลขทศนิยมแบบคิดเครื่องหมาย มีค่าเท่ากับ 2.5
```

### 1.3.3.6 double

คือชนิดข้อมูลที่เป็นตัวเลขทศนิยม 64 บิตแบบไม่คิดเครื่องหมาย มีค่า 0 ถึง $1.7976931348623157 \times 10^{+38}$

**ตัวอย่างที่ 1-12**

```
double y = 10.75;  // ประกาศตัวแปร y เป็นตัวแปรชนิดทศนิยมไม่คิดเครื่องหมาย มีค่าเท่ากับ 10.75
```

### 1.3.3.7 char

คือชนิดข้อมูลที่เป็นตัวอักษรเช่น A, d, และ $ ในรูปแบบ Unicode มีขนาด 1 ไบต์หรือ 8 บิต เขียนได้โดยใช้ ‘ หรืออัญประกาศเดี่ยวหรือซิงเกิลโคต (single quote) คร่อมข้อมูล ข้อมูลชนิด char จะถูกเข้ารหัสเป็นรหัสแอสกี้ (ASCII) มีค่าเป็นตัวเลขระหว่าง -128 ถึง 127

**ตัวอย่างที่ 1-13**

```
char x = 'a';              // ประกาศตัวแปร x เป็นตัวแปรชนิดตัวอักษร มีค่าเท่ากับรหัสแอสกี้ของอักษร a
```

### 1.3.3.8 unsigned char

คือชนิดข้อมูลที่เป็นตัวอักษรเช่นเดียวกับ char แต่ค่าของข้อมูลจะเป็นแบบไม่คิดเครื่องหมาย จึงมีค่าในช่วง 0 ถึง 255

**ตัวอย่างที่ 1-14**

```
unsigned char y = 'b';
unsigned char z = 200;      // ให้ค่าตัวแปร z มีค่าเริ่มต้นเป็น 200
```

### 1.3.3.9 byte

คือชนิดข้อมูลขนาด 8 บิต เก็บค่าเป็นตัวเลขระหว่าง 0 ถึง 255 ชนิดข้อมูลแบบนี้มักนำไปใช้กับตัวแปรสำหรับรับส่งข้อมูลจากพอร์ตอนุกรม (หรือ USB serial port : พอร์ต USB ที่ทำงานเป็นพอร์ตอนุกรมเสมือน) และแสดงตัวอักษรในรูปแบบที่ง่ายกว่าข้อมูลชนิด char

**ตัวอย่างที่ 1-15**

```
byte x  =  128;           // ประกาศตัวแปร x ที่มีขนาด 8 บิต มีค่าเริ่มต้นเท่ากับ 128
```

### 1.3.3.10 boolean

คือ ชนิดข้อมูลที่มีค่าเป็นจริงและเท็จ ปกติข้อมูลชนิด boolean จะใช้ควบคุมคำสั่งที่ใช้ในการตัดสินใจการทำงานของโปรแกรม ค่าของข้อมูลมี 2 ค่าคือ **จริง (true)** และ **เท็จ (false)**

**ตัวอย่างที่ 1-16**

```
boolean x = true;
boolean y = false;
```

## 1.4 อะเรย์ (Array)

**อะเรย์คือชุดของข้อมูล** โดยข้อมูลนั้นจะเป็นชนิดใดก็ได้ แต่ในอะเรย์ชุดเดียวกัน ชนิดของข้อมูลต้องเหมือนกันทั้งชุด ข้อมูลแต่ละค่าจะถูกเรียกด้วยเลขดัชนี (index) โดยข้อมูลแรกสุดคือ [0] ข้อมูลถัดมาคือ [1] และเรียกต่อกันไป

### 1.4.1 การประกาศตัวแปรอะเรย์

การประกาศตัวแปรอะเรย์ (array) คล้ายกับการประกาศตัวแปรทั่วไป แต่ต่างกันที่ array จะต้องมีเครื่องหมาย [ ] ต่อท้ายชื่อตัวแปรทุกครั้งเพื่อบอกว่าตัวแปรนี้เป็นตัวแปรชนิด array เรามักจะประกาศตัวแปร ชนิด array ไว้ด้านบนสุดของฟังก์ชั่นหรือตอนต้นของโปรแกรมเช่นเดียวกับตัวแปรทั่วไป และตัวแปรชนิด array ที่ถูกประกาศในฟังก์ชั่นหนึ่งจะไม่สามารถเรียกใช้ได้ในอีกฟังก์ชั่นหนึ่งเช่นกัน การกำหนดค่าให้ตัวแปรอาจทำพร้อมกับการประกาศตัวแปรก็ได้ โดยข้อมูลต้องอยู่ใน { } และแยกแต่ละข้อมูลด้วยเครื่องหมาย ,

**รูปแบบ** :

```
datatype var[element];
```

โดยที่ datatype คือ ชนิดข้อมูลที่จะให้ตัวแปรนั้นเก็บค่า
var คือ ชื่อของตัวแปร
element คือจำนวนข้อมูลที่เก็บในตัวแปร อาจไม่ระบุก็ได้

**ตัวอย่างที่ 1-17**

```
float data[5];
char string[];
char string1[7] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘!’, ‘\0’};    // ประกาศตัวแปรชื่อ string1 มีค่าเป็น hello!
char string2[] = “hello there!”;                          // ประกาศตัวแปรชื่อ string2 มีค่าเป็น hello there!
int numbers[] = { 90, 150, 30 };
// ประกาศตัวแปรชื่อ numbers ซึ่งเป็นตัวแปรชนิด array
// โดย numbers[0] มีค่าเป็น 90, numbers[1] มีค่าเป็น 150 และ numbers[2] มีค่าเป็น 30
```

จากตัวอย่างตัวแปรชนิด char สามารถกำหนดให้เป็นอะรเย์ได้ โดยเรียกว่า **ชุดตัวอักษร (ข้อความ)** หรือ**สายอักขระ**หรือ **string**

## 1.4.2 การอ่านค่าจากตัวแปรชนิดอะเรย์

การอ่านค่าจากตัวแปรชนิดนี้ต้องทำทีละตำแหน่ง ยกเว้นการอ่านค่าข้อมูลจากอะเรย์บางชนิด เช่น char สามารถอ่านได้หลายตำแหน่งพร้อมกัน

ถ้าตัวแปอะเรย์ชุดหนึ่งมี n ข้อมูล ตัวแปรชุดนี้จะมีค่าดัชนีหรือ index ตั้งแต่ 0 ถึง n-1 เช่น ตัวแปร numbers[3] จะมี 3 ข้อมูล และมี index เป็น 0 ถึง 2 หากต้องการอ่านค่าตัวแปรอะเรย์ ในตำแหน่งที่ n จะต้องแทนค่าของ index ด้วย n-1 โดย n คือขนาดของอะเรย์

**รูปแบบ :**

var[index]

โดยที่ var คือ ชื่อของตัวแปรอะเรย์

index คือ ตำแหน่งของตัวแปรในชุดตัวแปรอะเรย์ที่ต้องการอ่านข้อมูล

**ตัวอย่างที่ 1-18**

```
int numbers[3] = { 90, 150, 30 };   // ประกาศตัวแปรชื่อ numbers เป็นตัวแปรอะเรย์ 3 ชุด
int x= numbers[1];                  // ประกาศตัวแปร x และกำหนดให้มีค่าเท่ากับข้อมูลที่ 2 ของ
                                    // ตัวแปร numbers ซึ่งเท่ากับ 150 เนื่องจาก index คือ 1
```

## 1.4.3 การกำหนดค่าให้ตัวแปรอะเรย์

การกำหนดค่าให้ตัวแปรอะเรย์ทำได้โดยใช้เครื่องหมาย = เช่นเดียวกับตัวแปรทั่วไปซึ่งจะกำหนดค่าให้ได้ก็ต่อเมื่อชนิดข้อมูลและชนิดของตัวแปรเป็นชนิดเดียวกัน

**รูปแบบ :**

var[index] = value;

โดยที่ var คือ ชื่อของตัวแปรชนิด array

index คือ ตำแหน่งของตัวแปรในชุดตัวแปรชนิด array ที่ต้องการเก็บข้อมูล

value คือ ข้อมูลที่ต้องการเขียนหรือเก็บค่า

**ตัวอย่างที่ 1-19**

```
int numbers[3];             // ประกาศตัวแปรชื่อ numbers เป็นตัวแปรอะเรย์ 3 ชุด
void setup()
{
    numbers[0] = 90;        // กำหนดให้ตัวแปร numbers ชุด 0 มีค่าเท่ากับ 90
    numbers[1] = 150;       // กำหนดให้ตัวแปร numbers ชุด 1 มีค่าเท่ากับ 150
    numbers[2] = 30;        // กำหนดให้ตัวแปร numbers ชุด 2 มีค่าเท่ากับ 30
}
void loop()
{}
```

# 1.5 ฟังก์ชั่นในซอฟต์แวร์ Wiring

ในซอฟต์แวร์ Wiring นอกจากฟังก์ชั่นหลัก 2 ตัวคือ setup() และ loop() แล้ว ผู้พัฒนาโปรแกรมสามารถสร้างฟังก์ชั่นเพิ่มเติมขึ้นมาเพื่อใช้งานเองได้ โดยฟังก์ชั่นที่สร้างขึ้นใหม่สามารถเรียกใช้จากฟังก์ชั่น setup() และ loop() หรือจากฟังก์ชั่นใดๆ ก็ได้

ฟังก์ชั่นใน Wiring แบ่งออกเป็น 2 ประเภทคือ **ฟังก์ชั่นที่เมื่อทำงานจบแล้วมีการคืนค่า** และ **ฟังก์ชั่นที่เมื่อทำงานจบแล้วไม่คืนค่า (void)** เช่น setup() และ loop()

## 1.5.1 ฟังก์ชั่นแบบไม่คืนค่า : void

เป็นฟังก์ชั่นที่ไม่ต้องการตัวแปรมารองรับเวลาเรียกใช้ และจะไม่มีการใช้คำสั่ง return

**รูปแบบ** :

```
void  function(parameters)
{
    statements;
}
```

โดยที่ function คือ ชื่อของฟังก์ชั่น ใช้หลักการตั้งชื่อเหมือนกับการตั้งชื่อตัวแปร

parameters คือ ข้อมูลที่ทำการส่งผ่านระหว่างฟังก์ชั่น ในบางฟังก์ชั่นอาจไม่มีพารามิเตอร์ก็ได้หรืออาจมีหลายพารามิเตอร์ โดยแยกแต่ละพารามิเตอร์ด้วยเครื่องหมาย , ในการประกาศฟังก์ชั่นในส่วนของพารามิเตอร์เหมือนกับการประกาศตัวแปร ฟังก์ชั่นที่มีพารามิเตอร์เวลาเรียกใช้ต้องใส่ค่าของพารามิเตอร์ให้ถูกต้องด้วยทุกครั้ง จำนวนและตำแหน่งของพารามิเตอร์ในตอนรับและส่งต้องตรงกัน

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

<u>**ตัวอย่างที่ 1-20**</u>

```
void setup()
{
    pinMode(17, OUTPUT);
}
void loop()
{
    digitalWrite(17, HIGH);
}
```

## 1.5.2 คำสั่งหลักของฟังก์ชั่นแบบคืนค่า : return

return คือ คำสั่งที่ใช้ในการส่งข้อมูลจากฟังก์ชั่นหลังจากจบการทำงาน โดยข้อมูลที่ส่งกลับนั้นต้องเป็นชนิดเดียวกับฟังก์ชั่นที่ประกาศ ถ้าหากประกาศฟังก์ชั่นเป็นชนิด void (ฟังก์ชั่นชนิดไม่คืนค่า) จะไม่สามารถใช้คำสั่ง return ในการส่งค่าได้ นอกจากนี้คำสั่ง return ยังใช้ในการหยุดการทำงานของฟังก์ชั่นเพื่อไม่ให้โปรแกรมทำงานในส่วนที่เหลือแล้วกลับมายังตำแหน่งที่เรียกใช้งานฟังก์ชั่น

**รูปแบบ** :

```
type function(parameters) {
    statements;
    return value;
}
```

โดยที่ type คือ ชนิดของฟังก์ชั่นซึ่งดูจากชนิดข้อมูลที่ทำการ return ในที่นี้คือ value ชนิดของฟังก์ชั่นจึงเหมือนกับชนิดตัวแปร

function คือ ชื่อของฟังก์ชั่นใช้หลักการตั้งชื่อเหมือนกับการตั้งชื่อตัวแปร

parameters คือ ข้อมูลที่ทำการส่งผ่านระหว่างฟังก์ชั่น

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

value คือ ข้อมูลที่ฟังก์ชั่นจะทำการคืนค่าหลังจากจบการทำงาน

<u>**ตัวอย่างที่ 1-21**</u>

```
int val = 30;                    //global declaretion
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    int t = timestwo(val);
    Serial.println(t);
}
int timestwo(int dVal)
{
    dVal = dVal * 2;
    return dVal;
}
```

จากตัวอย่างฟังก์ชั่น timestwo เป็นฟังก์ชั่นที่คืนค่าเป็นเลขจำนวนเต็ม มีพารามิเตอร์คือ dVal ซึ่งเป็นตัวแปรชนิด int ดังนั้นเมื่อเรียกใช้ฟังก์ชั่น timestwo ที่ฟังก์ชั่นloop() จึงมีการส่งข้อมูลที่เป็นชนิด int มาด้วย

ตัวแปรที่ทำการประกาศในฟังก์ชั่นหนึ่งจะไม่สามารถใช้งานได้ในอีกฟังก์ชั่นหนึ่ง หากต้องการให้ตัวแปรใช้งานได้ในทั้งสองฟังก์ชั่น ทำได้โดยประกาศตัวแปรเป็น global นั่นคือ การประกาศตัวแปรนอกฟังก์ชั่นใดๆ เช่น การประกาศ val ดังในตัวอย่างที่ 1-21

## 1.6 การทำซ้ำ(Iteration)

การกระทำคำสั่งต่างๆ (statement) ภายในบล็อก {} ซ้ำๆ อย่างมีเงื่อนไข จะเรียกว่า ***ลูป (loop)*** คำสั่งที่ใช้เพื่อการวนลูปคือ **for()** และ **while()**

### 1.6.1 while()

while() เป็นคำสั่งที่ใช้สร้างลูป การทำงานของ while() จะกระทำคำสั่งต่างๆ หรือ statement ต่อเนื่องไปเรื่อยๆ และวนทำซ้ำต่อเนื่องจนกว่าจะพบการตรวจสอบเงื่อนไข (expression) แล้วเป็นจริง ควรจะมีการกำหนดค่าเวลาหรือจำนวนของการวนลูป มิเช่นนั้นอาจทำให้โปรแกรมจะไม่มีวันออกจากลูป while()

**รูปแบบ** :

```
while(expression) {
    statements;
}
```

โดยที่ expression คือ เงื่อนไขในการตรวจสอบว่าจะทำในส่วนของสเตตเมนต์ (statement) หรือไม่

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

**<u>ตัวอย่างที่ 1-22</u>**

```
int i = 0;
void setup()
{
    while(i<5)
    {
        i++;
    }
    j=i;        // j จะมีค่าเป็น 4
}
void loop()
{}
```

การใช้ลูป while() ควรระวังเรื่องเงื่อนไข เพราะอาจทำให้โปรแกรมไม่มีวันออกจากลูป (break out) ***ดังนั้นลูป while() เหมาะสำหรับการวนลูปที่ไม่รู้จำนวนรอบแน่ชัด***

## 1.6.2 for()

for() เป็นคำสั่งที่ใช้สร้างลูป โครงสร้างของคำสั่ง for() มี 3 ส่วนคือ ค่าเริ่มต้น (init), ตรวจสอบ (test) และปรับปรุง (update) แต่ละส่วนแบ่งแยกกันด้วยเซมิโคลอน (semi-colon) ; ลูปจะทำซ้ำต่อเนื่องจนกว่าผลการตรวจสอบจะออกมาเป็นเท็จ ขั้นตอนการทำงานของลูป for() สรุปได้ดังนี้

(1) ทำงานในส่วนของค่าเริ่มต้น (init statement)

(2) ตรวจสอบค่า (test) ว่า เป็นจริงหรือเท็จ

(3) ถ้าเป็นจริง จะทำงานต่อในขั้นตอน (4) แต่ถ้าเป็นเท็จจะทำงานในขั้นตอน (6)

(4) ทำงานในส่วนของสเตตเมนต์ภายในบล็อก {}

(5) ทำงานในส่วนของการรปรับปรุงค่า (update) แล้วกลับไปทำงานในขั้นตอน (2) อีกครั้ง

(6) ออกจากลูป

**รูปแบบ** :

```
for(init; test; update)
{
    statements;
}
```

**โดยที่** expression คือ การกระทำระหว่างตัวดำเนินการ (operator) และตัวถูกดำเนินการ(operand)

init คือ สเตตเมนต์ที่ทำงานในตอนเริ่มต้นของลูป

test คือ ถ้าผลตรวจสอบออกมาเป็นจริง จะทำงานในส่วนของ statements

update คือ คำสั่งที่ต้องกระทำเมื่อวนลูปครบหนึ่งรอบ

statements คือ คำสั่งที่ต้องกระทำ เมื่อผลการตรวจสอบออกมาเป็นจริงในแต่ละรอบ

<u>**ตัวอย่างที่** 1-23</u>

```
int i;
void setup()
{
    for(i=0; i<10; i=i+1)
    {
        digitalWrite(i, HIGH);
    }
}
void loop()
{}
```

***ดังนั้นลูป loop for() จึงเหมาะสำหรับการวนทำงานที่ทราบจำนวนรอบอย่างแน่ชัด***

### 1.6.3 break

**break เป็นคำสั่งที่ใช้จบการทำงานของลูป** เช่น switch(), for(), และ while() จากนั้นจะข้ามไปกระทำคำสั่งที่ถัดไปจากนั้น break เป็นคำสั่งที่หยุดการทำงานของลูปต่างๆ เช่น for() และ while() แม้ว่าเงื่อนไขของลูปจะยังเป็นจริงอยู่ก็ตาม

**รูปแบบ** :

```
break;
```

**<u>ตัวอย่างที่ 1-24</u>**

```
int i=0;
int j;
void setup()
{
    while(i<5)
    {
        i++;
        break;
    }
    j=i;            // j จะมีค่าเป็น 1
}
void loop()
{}
```

นอกจากนั้น break ยังเป็นส่วนหนึ่งของคำสั่ง switch() ซึ่งจะอธิบายในส่วนถัดไป

# 1.7 เงื่อนไข (Conditionals)

การตัดสินใจของโปรแกรมเกิดจากการตรวจสอบเงื่อนไข ฟังก์ชั่นที่ใช้ในการตัดสินใจหรือตรวจสอบเงื่อนไขประกอบด้วย switch() และ if()

## 1.7.1 if...else

**if() เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไข (expression)** ถ้าหากว่าตรวจสอบเงื่อนไขแล้วเป็นจริงจะทำคำสั่งต่างๆ (statement) ที่อยู่ในบล็อก {} ของ if() หากเป็นเท็จจะไม่ทำคำสั่งที่อยู่ในบล็อก {} ของ if() ถ้าหากมีคำสั่ง else ร่วมด้วย โปรแกรมจะเลือกทำคำสั่งต่างๆที่อยู่ในบล็อก {} ของ else โปรแกรมของคุณอาจมีหรือไม่มีในส่วนของ else ก็ได้ ขึ้นอยู่กับจำนวนของเงื่อนไขที่ต้องการตรวจสอบ

**รูปแบบ** :

```
if(expression) {
    statements;
} else {
    statements;
}
```

โดยที่ expression คือ เงื่อนไขในการตรวจสอบว่า จะให้กระทำคำสั่งในส่วนของ statement หรือไม่

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

**ข้อควรรู้** : else ต้องใช้ร่วมกับ if() เสมอ แต่ if() ไม่จำเป็นต้องมี else ก็ได้

หากต้องการตรวจสอบมากกว่าหนึ่งเงื่อนไข ให้ใช้คำสั่ง **else if()** เพื่อเพิ่มเงื่อนไขอื่นๆ อาจจะมีการใช้ else if() มากกว่า1 ครั้งได้

เมื่อต้องการใช้คำสั่ง else จะต้องกำหนดไว้อยู่ท้ายสุด และโปรแกรมจะทำงานในส่วนของ else ก็ต่อเมื่อตรวจสอบเงื่อนไขทั้งหมดแล้วพบว่า เป็นเท็จทั้งหมด ดังนี้

**รูปแบบ** :

```
if(expression) {
    statements;
} else if(expression) {
    statements;
} else {
    statements;
}
```

โดยที่ expression คือ เงื่อนไขในการตรวจสอบว่าจะกระทำคำสั่งในส่วนของ statement หรือไม่

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

**ตัวอย่างที่ 1-25**

```
char letter = 'b';
void setup()
{
    Serial.begin(9600);            // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(letter=='a')
    {
        Serial.println("Alpha");   // โปรแกรมจะไม่ทำงานในส่วนนี้
    }
    else if(letter=='b')
    {
        Serial.println("Bravo");   // แสดงข้อความ Bravo ใน Serial Monitor ทำงาน
    }
    else
    {
        Serial.println("None");    // โปรแกรมจะไม่ทำงานในส่วนนี้
    }
}
void loop()
{}
```

### 1.7.2 switch

switch() เป็นคำสั่งที่ใช้ในการตัดสินใจเช่นเดียวกับ if() ***โดย switch() เหมาะสำหรับการตัดสินใจที่มีทางเลือก 3 ทางขึ้นไป*** โดยตัดสินใจจากการตรวจสอบข้อมูลในส่วนเงื่อนไข (expression) ว่าตรงกับข้อกำหนด (label) ในกรณี (case) ใด จากนั้นจะกระโดดไปทำงานในส่วนนั้นทันที แต่ละ case ถูกแบ่งออกจากกันด้วยคำสั่ง break ดังนั้นเมื่อจบการทำงานใน caseนั้นๆ แล้วจะออกจาก switch() ทันที ไม่ทำ caseอื่นๆ อีก

เมื่อมีการใช้คำสั่ง switch() โปรแกรมจะตรวจสอบตั้งแต่ case แรกจนถึง case สุดท้าย หากข้อมูลในส่วนเงื่อนไขไม่ตรงกับค่าใน case ใดๆ และถ้าโปรแกรมมีการกำหนดการทำงานในขั้นต้น หรือ default โปรแกรมจะทำงานในส่วนนั้นแทน โดยในส่วนของ default จะมีหรือไม่มีก็ได้

**รูปแบบ** :

```
switch(expression)
{
    case label:
    statements;
    break;
    default:
    statements;
}
```

โดยที่ expression คือ ข้อมูลหรือตัวแปรที่ใช้เป็นเงื่อนไขในการตรวจสอบว่าตรงกับข้อกำหนดหรือ label ในกรณีหรือ caseใด ข้อมูลหรือตัวแปรที่ใช้ต้องเป็นข้อมูลชนิด byte, char หรือ int

label คือ ข้อมูลที่ใช้ในการเปรียบเทียบ ควรเป็นข้อมูลชนิดเดียวกับข้อมูลหรือตัวแปรที่ใช้เป็นเงื่อนไข

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

**ตัวอย่างที่ 1-26**

```
char letter = 'b';
void setup()
{
    Serial.begin(9600);             // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    switch(letter)
    {
        case 'a':
        case 'A':
            Serial.println("Alpha");   // โปรแกรมจะไม่ทำงานในส่วนนี้
            break;
        case 'b':
        case 'B':
            Serial.println("Bravo");   // แสดงข้อความ Bravo ใน Serial Monitor ทำงาน
            break;
        default:
            Serial.println("None");    // โปรแกรมจะไม่ทำงานในส่วนนี้
            break;
    }
}
void loop()  {
}
```

จากตัวอย่าง จะเห็นได้ว่า คำสั่ง case สามารถตรวจสอบได้มากกว่าหนึ่งค่าพร้อมๆ กัน ในที่นี้ คือ 'a' และ 'A' เพราะในการเขียนโปรแกรมภาษา C/C++ นั้นเป็นแบบ case-sensitive คือ ให้ความสนใจในตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก

## 1.8 ตัวดำเนินการทางคณิตศาสตร์( Mathemetic Operators)

ตัวดำเนินการทางคณิตศาสตร์ใช้การคำนวณทางคณิตศาสตร์ในโปรแกรมภาษา C/C++ ของซอฟต์แวร์ Wiring มีดังนี้

| สัญลักษณ์ | ชื่อ |
|---|---|
| + | บวก (addition) |
| - | ลบ (minus) |
| % | หารเอาเศษ (modulo) |
| += | บวกแล้วเก็บค่าไว้ที่ตัวตั้ง(add assign) |
| = | ลบแล้วเก็บค่าไว้ที่ตัวตั้ง(subtract assign) |
| / | หาร (divide) |
| * | คูณ (multiply) |
| ++ | เพิ่มค่า (increment) |
| -- | ลดค่า (decrement) |

### 1.8.1 + บวกค่า (Addition)

**รูปแบบ :**

```
value1 + value2
```

โดยที่ value1 เป็นข้อมูลชนิด int หรือ float
value2 เป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 1-27**

```
int a = 50 + 5;     // ค่าของ a คือ 55
int b = a + 5;      // ค่าของ b คือ 60
```

### 1.8.2 - ลบค่า (minus)

**รูปแบบ :**

```
value1 - value2
```

โดยที่ value1 เป็นข้อมูลชนิด int หรือ float
value2 เป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 1-28**

```
int c = 50 - 5;     // ค่าของ c คือ 45
int d = c - 5;      // ค่าของ d คือ 40
int e = d - 60;     // ค่าของ e คือ -20
```

### 1.8.3 * คูณค่า (multiply)

**รูปแบบ :**

value1 * value2

โดยที่ value1 และ value2 เป็นข้อมูลชนิด int, float, byte หรือ char

**ตัวอย่างที่ 1-29**

```
int e = 50 * 5;          // ค่าของ e คือ 250
int f = e * 5;           // ค่าของ f คือ 1250
```

### 1.8.4 / หารค่า (divide)

**รูปแบบ :**

value1 / value2

โดยที่ value1 เป็นข้อมูลชนิด int หรือ float

value2 เป็นข้อมูลชนิด int หรือ float แต่ต้องไม่เท่ากับ 0

**ตัวอย่างที่ 1-30**

```
int g = 50 / 5;          // ค่าของ g คือ 10
int h = g / 5;           // ค่าของ h คือ 2
```

### 1.8.5 % คำนวณหาเศษของการหาร (modulus)

**รูปแบบ :**

value1 % value2

โดยที่ value1 และ value2 เป็นข้อมูลชนิด int

**ตัวอย่างที่ 1-31**

```
int a = 20%100;          // ค่าของ a คือ 20
int b = 75%100;          // ค่าของ b คือ 75
```

### 1.8.6 += บวกแล้วเก็บค่าไว้ที่ตัวตั้ง (add assign)

ลักษณะการทำงานของคำสั่งนี้คือ a+=b มีค่าเหมือนกับ a = a + b

**รูปแบบ :**

value1 += value2;

โดยที่ value1 เป็นข้อมูลชนิด int หรือ float

value2 สามารถเป็นข้อมูลชนิดใดก็ได้ แต่ต้องเป็นชนิดเดียวกับ value1

**ตัวอย่างที่ 1-32**

```
int a = 50;
int b = 23;
a += b;                  // ค่าของ a คือ 73
```

### 1.8.7 -= ลบแล้วเก็บค่าไว้ที่ตัวตั้ง (subtract assign)

ลักษณะการทำงานของคำสั่งนี้คือ a-=b มีค่าเหมือนกับ a = a - b

**รูปแบบ :**

```
value1 -= value2;
```

โดยที่ value1 และ value2 เป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 1-33**

```
int a = 50;
int b = 23;
a -= b; // ค่าของ a คือ 27
```

### 1.8.8 ++ เพิ่มค่า (increment)

เป็นคำสั่งเพิ่มค่าตัวแปรขึ้นหนึ่งค่า ลักษณะการทำงานคือ a++ มีค่าเหมือนกับ a = a + 1

**รูปแบบ :**

```
value++
```

โดยที่ value เป็นข้อมูลชนิด int

**ตัวอย่างที่ 1-34**

```
int a = 0;        // ค่าของ a คือ 0
a++;
int b = a;        // ค่าของ b คือ 1
```

### 1.8.9 -- ลดค่า (decrement)

เป็นคำสั่งลดหรือลบค่าตัวแปรลงหนึ่ง มีลักษณะการทำงานคือ a-- มีค่าเหมือนกับ a = a - 1

**รูปแบบ :**

```
value --
```

โดยที่ value เป็นข้อมูลชนิด int

**ตัวอย่างที่ 1-35**

```
int a = 5;        // ค่าของ a คือ 5
a--;
int b = a;        // ค่าของ b คือ 4
```

## 1.9 ตัวดำเนินการเชิงเปรียบเทียบ (Relational Operators)

ตัวดำเนินการเชิงเปรียบเทียบใช้ในการเปรียบเทียบค่าในซอฟต์แวร์ Wiring มีดังนี้

| สัญลักษณ์ | ชื่อ |
|---|---|
| < | น้อยกว่า (less than) |
| >= | มากกว่าหรือเท่ากับ (greater than or equal to) |
| == | เท่ากัน (equality) |
| <= | น้อยกว่าหรือเท่ากับ (less than or equal to) |
| > | มากกว่า (greater than) |
| != | ไม่เท่ากัน (inequality) |

โดยให้ผลการทำงานเป็น **จริง (true)** หรือ **เท็จ (false)**

### 1.9.1 < น้อยกว่า (less than)

เป็นคำสั่งเปรียบค่าน้อยกว่า *หากใช่* จะคืนค่า เป็นจริง

**รูปแบบ** :

value1 < value2

**โดยที่** value1 และ value2 เป็นข้อมูลชนิด int หรือ float

**<u>ตัวอย่างที่ 1-36</u>**

```
int a = 22;
int b = 23;
void setup()
{
    Serial.begin(9600);       // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(a < b)
    {
        Serial.print("variable a is less then variable b ");
                              // แสดงข้อความที่หน้าต่าง Serial Monitor เมื่อค่าของ a น้อยกว่า b
    }
}
void loop()
{}
```

## 1.9.2 <= น้อยกว่าหรือเท่ากับ (less than or equal to)

เป็นคำสั่งเปรียบเทียบค่าน้อยกว่าหรือเท่ากับระหว่างค่าของตัวแปรสองตัว

**รูปแบบ :**

value1 <= value2

**โดยที่** value1 และ value2 เป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 1-37**

```
int a = 22;
int b = 23;
void setup()
{
    Serial.begin(9600);      // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(a <= b)
    {
        Serial.print("variable a is less or equal to variable b ");
        // แสดงข้อความที่หน้าต่าง Serial Monitor เมื่อค่าของ a น้อยกว่าหรือเท่ากับ b
    }
}
void loop()
{}
```

## 1.9.3 > มากกว่า (greater than)

เป็นคำสั่งเปรียบเทียบค่ามากกว่าระหว่างค่าของตัวแปรสองตัว

**รูปแบบ :**

value1 > value2

**โดยที่** value1 และ value2 เป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 1-38**

```
int a = 5;
int b = 13;
void setup()
{
    Serial.begin(9600);      // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(b > a)
    {
        Serial.print("variable b is larger the variable a");
        // แสดงข้อความที่หน้าต่าง Serial Monitor เมื่อค่าของ a มากกว่า b
    }
}
void loop()
{}
```

## 1.9.4 >= มากกว่าหรือเท่ากับ (greater than or equal to)

เป็นคำสั่งเปรียบเทียบค่าน้อยกว่าเท่ากับระหว่างค่าของตัวแปรสองตัว

**รูปแบบ :**

value1 >= value2

โดยที่ value1 และ value2 เป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 1-39**

```
int a = 23;
int b = 23;
void setup()
{
    Serial.begin(9600);          // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(a >= b)
    {
        Serial.print("variable a is greater or equal to variable b ")
        // แสดงข้อความที่หน้าต่าง Serial Monitor เมื่อค่าของ a มากกว่าหรือเท่ากับ b
    }
}
void loop()
{}
```

## 1.9.5 = = เท่ากับ (equality)

เป็นคำสั่งเปรียบเทียบค่าเท่ากับระหว่างค่าของตัวแปรสองตัว

**รูปแบบ :**

value1 == value2

โดยที่ value1 และ value2 เป็นข้อมูลชนิด int, float, char, byte, boolean

**ตัวอย่างที่ 1-40**

```
int a = 23;
int b = 23;
void setup()
{
    Serial.begin(9600);          // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(a == b)
    {
        Serial.print("variables a and b are equal");
        // แสดงข้อความที่หน้าต่าง Serial Monitor เมื่อค่าของ a เท่ากับ b
    }
}
void loop()
{}
```

## 1.9.6 != ไม่เท่ากัน (inequality)

เป็นคำสั่งเปรียบเทียบค่าไม่เท่ากันของตัวแปรสองตัว

**รูปแบบ :**

value1 != value2

**โดยที่** value1 และ value2 เป็นข้อมูลชนิด int, float, char, byte, boolean

**ตัวอย่างที่ 1-41**

```
int a = 22;
int b = 23;
void setup()
{
    Serial.begin(9600);          // ตั้งค่าอัตราบอดให้กับการสื่อสารข้อมูลอนุกรมเป็น 9600
    if(a != b)
    {
        Serial.print("variable a is not equal to variable b");
        // แสดงข้อความที่หน้าต่าง Serial Monitor เมื่อค่าของ a ไม่เท่ากับ b
    }
}
void loop()
{}
```

## 1.10 ตัวดำเนินการทางตรรกะ (Logical operators)

ตัวดำเนินทางตรรกะใช้การคำนวณทางตรรกะ เพื่อตรวจสอบค่าของตัวแปรสองค่า โดยตัวดำเนินการทางตรรกะในซอฟต์แวร์ Wiring มีดังนี้

| สัญลักษณ์ | ชื่อ |
|---|---|
| \|\| | การออร์ (logical OR) |
| && | การแอนด์ (logical AND) |
| ! | การนอต (logical NOT) |

### 1.10.1 || การออร์ (logical OR)

ใช้เปรียบเทียบค่า 2 ค่า โดยให้ผลลัพธ์เป็นค่า true เมื่อค่าของตัวแปรตัวใดตัวหนึ่งเป็นจริง ดังนี้

| ตัวแปร 1 | ตัวดำเนินการ | ตัวแปร 2 | ผลลัพธ์ |
|---|---|---|---|
| true | \|\| | true | true |
| true | \|\| | false | true |
| false | \|\| | true | true |
| false | \|\| | false | false |

**รูปแบบ :**

expression1 || expression2

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ(operands)

**ตัวอย่างที่ 1-42**

```
boolean x = true;
boolean y = false;
boolean a,b,c,d;
void setup()
{
    a = x || y;         // a ได้ผลเป็น true
    b = y || x;         // b ได้ผลเป็น true
    c = x || x;         // c ได้ผลเป็น true
    d = y || y;         // d ได้ผลเป็น false
}
void loop()
{}
```

## 1.10.2 && การแอนด์ (logical AND)

ใช้เปรียบเทียบค่า 2 ค่า โดยให้ผลลัพธ์เป็นค่า true เมื่อค่าของตัวแปรทั้งสองเป็นจริง ดังนี้

| ตัวแปร 1 | ตัวดำเนินการ | ตัวแปร 2 | ผลลัพธ์ |
|---|---|---|---|
| true | && | true | true |
| true | && | false | false |
| false | && | true | false |
| false | && | false | false |

**รูปแบบ :**

expression1 && expression2

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands)

**ตัวอย่างที่ 1-43**

```
boolean x = true;
boolean y = false;
boolean a,b,c,d;
void setup()
{
    a = x && y;          // a ได้ผลเป็น false
    b = y && x;          // b ได้ผลเป็น false
    c = x && x;          // c ได้ผลเป็น true
    d = y && y;          // d ได้ผลเป็น false
}
void loop()
{}
```

## 1.10.3 ! กลับค่าข้อมูล (NOT)

ใช้กลับค่าของข้อมูลชนิด Boolean ของตัวแปร คืนค่าเป็น false ถ้าค่าของตัวแปรเป็น true และคืนค่าเป็น true ถ้าค่าของตัวแปรเป็น false ดังนี้

| ตัวดำเนินการ | ตัวแปร | ผลลัพธ์ |
|---|---|---|
| ! | true | false |
| ! | false | true |

**รูปแบบ :**

!expression

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ(operands)

**ตัวอย่างที่ 1-44**

```
boolean a = true;
boolean b = false;
boolean c,d;
void setup()
{
    c=!a;           //c มีค่าเป็น false
    d=!b;           //d มีค่าเป็น true
}
void loop()
{}
```

## 1.11 ตัวดำเนินการระดับบิต (Bitwise operators)

ตัวดำเนินการระดับบิตใช้ในการคำนวณข้อมูลระดับบิต โดยต้องแปลงค่าให้อยู่ในรูปของเลขฐานสองก่อน แล้วเริ่มคำนวณทีละบิตที่ตรงกันของสองข้อมูล

| สัญลักษณ์ | ชื่อ |
|---|---|
| \| | การออร์ระดับบิต (bitwise OR) |
| \|= | การออร์ระดับบิตแล้วเก็บไว้ที่ตัวตั้ง(bitwise OR and assign) |
| & | การแอนด์ระดับบิต (bitwise AND) |
| &= | การแอนด์ระดับบิตแล้วเก็บไว้ที่ตัวตั้ง (bitwise AND and assign) |
| ^ | การเอ็กคลูซีฟ-ออร์ระดับบิต (bitwise XOR) |
| ~ | กลับบิต (bitwise one's complement) |
| << | เลื่อนบิตไปทางซ้าย (bitwise bit shift left) |
| >> | เลื่อนบิตไปทางขวา (bitwise bit shift right) |

## 1.11.1 | การออร์ระดับบิต (bitwise OR)

เป็นคำสั่งเปรียบเทียบบิตข้อมูล 2 ค่า (expression) โดยคืนค่าเป็น “0” ถ้าบิตที่เปรียบเทียบทั้งสองมีค่าเป็น “0” ทั้งคู่ นอกนั้นจะคืนค่าเป็น “1” มีรูปแบบการประมวลผลดังนี้

| ตัวแปร 1 | ตัวดำเนินการ | ตัวแปร 2 | ผลลัพธ์ |
|---|---|---|---|
| 1 | \| | 1 | 1 |
| 1 | \| | 0 | 1 |
| 0 | \| | 1 | 1 |
| 0 | \| | 0 | 0 |

**รูปแบบ** :

expression1 | expression2

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**ตัวอย่างที่ 1-45**

```
unsigned int a = 60;     // a มีค่าเป็น 60 ซึ่งเท่ากับ 0011 1100 ฐานสอง
unsigned int b = 13;     // b มีค่าเป็น 13 ซึ่งเท่ากับ 0000 1101 ฐานสอง
unsigned int c = 0;
c = a | b;               // c มีค่าเป็น 61 ซึ่งเท่ากับ 0011 1101 ฐานสอง
```

## 1.11.2 |= การออร์ระดับบิตแล้วเก็บค่าไว้ที่ตัวตั้ง (bitwise OR and assign)

เป็นการออร์บิต 2 บิตแล้วนำผลการออร์นั้นกลับไปเก็บไว้ที่ตัวแปรทางซ้ายมือหรือตัวตั้ง

**รูปแบบ** :

expression1 |= expression2;

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**ตัวอย่าง 1-46**

```
unsigned int a = 60;     // a มีค่าเป็น 60 ซึ่งเท่ากับ 0011 1100 ฐานสอง
unsigned int b = 13;     // b มีค่าเป็น 13 ซึ่งเท่ากับ 0000 1101 ฐานสอง
a |= b;                  // a มีค่าเป็น 61 ซึ่งเท่ากับ 0011 1101 ฐานสอง
```

### 1.11.3 & การแอนด์ระดับบิต (bitwise AND)

เป็นคำสั่งที่ใช้ในการเปรียบเทียบบิตข้อมูล 2 ค่า (expression) โดยคืนค่าเป็น "1" ถ้าบิตที่เปรียบเทียบทั้งสองมีค่าเป็น "1" ทั้งคู่ นอกนั้นจะคืนค่าเป็น "0" มีรูปแบบการประมวลผลดังนี้

| ตัวแปร 1 | ตัวดำเนินการ | ตัวแปร 2 | ผลลัพธ์ |
|---|---|---|---|
| 1 | & | 1 | 1 |
| 1 | & | 0 | 0 |
| 0 | & | 1 | 0 |
| 0 | & | 0 | 0 |

**รูปแบบ :**

expression1 & expression2

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**ตัวอย่างที่ 1-47**

```
unsigned int a = 60;     // a มีค่าเป็น 60 ซึ่งเท่ากับ 0011 1100 ฐานสอง
unsigned int b = 13;     // b มีค่าเป็น 13 ซึ่งเท่ากับ 0000 1101 ฐานสอง
unsigned int c = 0;
c = a & b;               // c มีค่าเป็น 12 ซึ่งเท่ากับ 0000 1100 ฐานสอง
```

### 1.11.4 &= การแอนด์ระดับบิตแล้วเก็บค่าไว้ที่ตัวตั้ง (bitwise AND and assign)

เป็นการออร์บิต 2 บิตแล้วนำผลการออร์นั้นกลับไปเก็บไว้ที่ตัวแปรทางซ้ายมือหรือตัวตั้ง

ดังนั้น a&=b จึงมีค่าเหมือนกับ a=a&b

**รูปแบบ :**

expression1 &= expression2;

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**ตัวอย่างที่ 1-48**

```
unsigned int a = 60;     // a มีค่าเป็น 60 ซึ่งเท่ากับ 0011 1100 ฐานสอง
unsigned int b = 13;     // b มีค่าเป็น 13 ซึ่งเท่ากับ 0000 1101 ฐานสอง
a &= b;                  // a มีค่าเป็น 12 ซึ่งเท่ากับ 0000 1100 ฐานสอง
```

## 1.11.5 ^ การเอ็กคลูซีฟ-ออร์ระดับบิต (bitwise XOR)

เป็นคำสั่งที่ใช้ในการเปรียบเทียบบิตข้อมูล 2 ค่า (expression) โดยคืนค่าเป็น "1" ถ้าบิตที่เปรียบเทียบทั้งสองมีค่าต่างกันจะคืนค่าเป็น "1" แต่ถ้าหากบิตที่เปรียบเทียบมีค่าเหมือนกันจะคืนค่าเป็น "0" มีรูปแบบการประมวลผลดังนี้

| ตัวแปร 1 | ตัวดำเนินการ | ตัวแปร 2 | ผลลัพธ์ |
|---|---|---|---|
| 1 | ^ | 1 | 0 |
| 1 | ^ | 0 | 1 |
| 0 | ^ | 1 | 1 |
| 0 | ^ | 0 | 0 |

**รูปแบบ :**

expression1 ^ expression2

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**<u>ตัวอย่างที่ 1-49</u>**

```
unsigned int a = 60;      // a มีค่าเป็น 60 ซึ่งเท่ากับ 0011 1100 ฐานสอง
unsigned int b = 13;      //  มีค่าเป็น 13 ซึ่งเท่ากับ  0000 1101 ฐานสอง
unsigned int c = 0;
c = a ^ b;                //    c มีค่าเป็น 49 ซึ่งเท่ากับ 0011 0001 ฐานสอง
```

## 1.11.6 ~ การกลับบิต (bitwise one's complement)

เป็นคำสั่งที่ใช้ในการกลับค่าลอจิกของบิตจาก "0" เป็น "1" หรือจากบิต "1" เป็น "0" โดยจะกลับทีละบิตของข้อมูล (expression) มีรูปแบบการประมวลผลดังนี้

| ตัวดำเนินการ | ตัวแปร | ผลลัพธ์ |
|---|---|---|
| ~ | 1 | 0 |
| ~ | 0 | 1 |

**รูปแบบ :**

~ expression

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**<u>ตัวอย่างที่ 1-50</u>**

```
unsigned int value=4;     // value มีค่าเป็น 4 ซึ่งเท่ากับ 0000 0100 ฐานสอง
value = ~ value;          // value มีค่าเป็น 251ซึ่งเท่ากับ 1111 1011 ฐานสอง
```

## 1.11.7 << และ <<= การเลื่อนบิตไปทางซ้าย

เป็นคำสั่งที่ใช้ในการเลื่อนบิตของข้อมูลของตัวแปรหรือ expression1 ไปทางซ้ายทีละบิตตามจำนวนครั้ง ซึ่งกำหนดโดยค่าตัวแปรหรือ expression 2 ที่อยู่ทางขวามือ และเพิ่มค่า 0 ให้กับบิตทางขวาสุดหรือบิต LSB (Least Significat Bit) ในทุกครั้งที่เลื่อนบิตไปทางซ้าย

**รูปแบบ** :

expression1 << expression2

เลื่อนบิตไปทางซ้ายตามจำนวนที่กำหนดด้วยค่าของ expression2

หรือ

expression1 <<= expression2;

เลื่อนบิตไปทางซ้ายตามจำนวนที่กำหนดด้วยค่าของ expression2 แล้วเก็บผลลัพธ์ลงในตัวตั้งหรือตัวแปรด้านซ้ายมือ

นั่นคือ value <<= shift; มีการทำงานเหมือนกับ value = value << shift;

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**<u>ตัวอย่างที่ 1-51</u>**

```
unsigned int value=4;      // value มีค่าเป็น  4 ซึ่งเท่ากับ 0000 0100 ฐานสอง
unsigned int shift=2;
value = value << shift;    // value มีค่าเป็น 16 ซึ่งเท่ากับ 0001 0000 ฐานสอง
value <<= shift;           // value มีค่าเป็น 64 ซึ่งเท่ากับ 0100 0000 ฐานสอง
```

## 1.11.8 >> และ >>= การเลื่อนบิตไปทางขวา (bitwise bit shift right)

เป็นคำสั่งที่ใช้ในการเลื่อนบิตของข้อมูลของตัวแปรหรือ expression1 ไปทางขวาทีละบิตตามจำนวนครั้ง ซึ่งกำหนดโดยค่าตัวแปรหรือ expression 2 ที่อยู่ทางขวามือ และเพิ่มค่า 0 ให้กับบิตทางซ้ายสุดหรือบิต MSB (Most Significat Bit) ในทุกครั้งที่เลื่อนบิตไปทางขวา

**รูปแบบ** :

expression1 >> expression2

เลื่อนบิตไปทางขวาตามจำนวนที่กำหนดด้วยค่าของ expression2

หรือ

expression1 >>= expression2;

เลื่อนบิตไปทางขวาตามจำนวนที่กำหนดด้วยค่าของ expression2 แล้วเก็บผลลัพธ์ลงในตัวตั้งหรือตัวแปรด้านซ้ายมือ

นั่นคือ value >>= shift; มีการทำงานเหมือนกับ value = value >> shift;

โดยที่ expression คือ การกระทำระหว่างตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) หรืออาจเป็นข้อมูล

**<u>ตัวอย่างที่ 1-52</u>**

```
unsigned int value=4;        // value มีค่าเป็น 4 ซึ่งเท่ากับ 0000 0100 ฐานสอง
unsigned int shift=1;
value = value >> shift;      // value มีค่าเป็น 2 ซึ่งเท่ากับ 0000 0010 ฐานสอง
value >>= shift;             // value มีค่าเป็น 1 ซึ่งเท่ากับ 0000 0001 ฐานสอง
```

# 1.12 ค่าคงที่ใน Wiring

ใน wiring มีค่าคงที่ต่างๆ เพื่อช่วยอำนวยความสะดวกในการคำนวณทางคณิตศาสตร์ ซึ่งคุณควรทำความรู้จักก่อน จะเริ่มเขียนโปรแกรมดังต่อไปนี้

| ค่าคงที่ | คำอธิบาย |
|---|---|
| PI | ค่าคงที่ทางคณิตศาสตร์ มีค่าประมาณ 3.14159 ใช้ร่วมกับฟังก์ชั่นในการคำนวณตรีโกณมิติ |
| HALF_PI | ค่าคงที่ทางคณิตศาสตร์ มีค่าประมาณ 1.57079 ซึ่งก็คือ ครึ่งหนึ่งของค่า PI ใช้ร่วมกับฟังก์ชั่นในการคำนวณตรีโกณมิติ |
| TWO_PI | ค่าคงที่ทางคณิตศาสตร์ มีค่าประมาณ 6.28318 ซึ่งก็คือสองเท่าของค่า PI ใช้ร่วมกับฟังก์ชั่นคำนวณตรีโกณมิติ |

**หมายเหตุ** : *ชื่อของค่าคงที่เหล่านี้เป็นคำสงวน (reserved word) นำไปตั้งเป็นชื่อตัวแปรไม่ได้*

# 1.13 คำสั่งเพิ่มเติมใน Wiring ที่ #define ไว้

นอกเหนือไปจากคำสั่งโครงสร้าง (structure) หลักที่เป็นไปตามมาตรฐานของโปรแกรมภาษา C/C++ ใน Wiring ยังมีคำสั่งพิเศษเพิ่มเติมอีกหลายคำสั่ง ดังนี้

## 1.13.1 null

ใช้กำหนดเป็นค่าว่าง หรือหมายถึง ตัวอักษร “ว่าง” ก็ได้ เขียนได้ทั้ง null และ NULL

**<u>ตัวอย่างที่ 1-53</u>**

```
char str[] = “some characters”;
if( str != null)
{
    Serial.print(c);            // พิมพ์ข้อความ some characters
}
```

## 1.13.2 false

ใช้กำหนดค่าลอจิกเมื่อเป็นเท็จ หรือเท่ากับ "0" ใช้กับตัวแปรชนิด boolean เท่านั้น

**ตัวอย่างที่ 1-54**

```
boolean b = false;
if(b == false)              // ถ้าค่าของ b เป็นเท็จ
{
    digitalWrite(17, HIGH); // ทำให้ขาพอร์ต 17 เป็นลอจิก "1" ถ้าค่าลอจิกของ b เป็นเท็จ
}
else
{
    digitalWrite(17, LOW);
}
```

## 1.13.3 true

ใช้กำหนดค่าลอจิกเมื่อเป็นจริง หรือมีค่าเท่ากับ "1" ใช้กับตัวแปรชนิด boolean เท่านั้น

**ตัวอย่างที่ 1-55**

```
boolean b = true;
if(b == true)               // ถ้าค่าของ b เป็นจริง
{
    digitalWrite(17, HIGH); // ทำให้ขาพอร์ต 17 เป็นลอจิก "1" ถ้าค่าลอจิกของ b เป็นจริง
}
else
{
    digitalWrite(17, LOW);
}
```

## 1.13.4 INPUT

เป็นคำสั่งกำหนดให้ขาพอร์ตเป็นอินพุตดิจิตอล โดยเขียนค่า "0" ไปยังรีจิสเตอร์กำหนดทิศทาง

**ตัวอย่างที่ 1-56**

```
pinMode(16, INPUT);     // กำหนดให้ขาพอร์ตหมายเลข 16 เป็นอินพุตดิจิตอล
```

## 1.13.5 OUTPUT

เป็นคำสั่งกำหนดให้ขาพอร์ตเป็นเอาต์พุตดิจิตอล โดยเขียนค่า "1" ไปยังรีจิสเตอร์กำหนดทิศทาง

**ตัวอย่างที่ 1-57**

```
pinMode(17, OUTPUT);    // กำหนดให้ขาพอร์ตหมายเลข 17 เป็นเอาต์พุตดิจิตอล
```

## 1.13.6 LOW

เป็นคำสั่งกำหนดค่าตัวแปรเป็นลอจิกต่ำ หรือเท่ากับ “0” ซึ่งมีระดับแรงดันเท่ากับ 0V และอาจหมายถึงสถานะปิดหรือ OFF ก็ได้

**<u>ตัวอย่างที่ 1-58</u>**

```
digitalWrite(17, LOW);      // เขียนค่าลอจิกต่ำไปยังขาพอร์ต 17
```

## 1.13.7 HIGH

เป็นคำสั่งกำหนดค่าตัวแปรเป็นลอจิกสูง หรือเท่ากับ “1” ซึ่งมีระดับแรงดัน +5V อาจหมายถึงสถานะ ON ก็ได้

**<u>ตัวอย่างที่ 1-59</u>**

```
digitalWrite(17, HIGH);     // เขียนค่าลอจิกสูงไปยังขาพอร์ต 17
```

## 1.13.8 CHANGE

เป็นคำสั่งกำหนดให้เกิดการอินเตอร์รัปต์ขึ้น เมื่อเกิดการเปลี่ยนแปลงลอจิกที่ขาอินพุตอินเตอร์รัปต์ภายนอก

**<u>ตัวอย่างที่ 1-60</u>**

```
attachInterrupt(0, myFunction, CHANGE);
// กำหนดให้ไปทำงานที่ myfunction เมื่อเกิดอินเตอร์รัปต์ที่ขาอินพุตอินเตอร์รัปต์ภายนอกช่อง 0
```

## 1.13.9 FALLING

เป็นคำสั่งกำหนดให้เกิดอินเตอร์รัปต์ขึ้น เมื่อเกิดการเปลี่ยนแปลงลอจิกสูงเป็นต่ำที่ขาอินพุตอินเตอร์รัปต์ภายนอก

**<u>ตัวอย่างที่ 1-61</u>**

```
attachInterrupt(0, myFunction, FALLING);
// ไปทำงานที่ myfunction เมื่อเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนลอจิกจากสูงเป็นต่ำขึ้น
// ที่ขาอินพุตอินเตอร์รัปต์ภายนอกช่อง 0
```

### 1.13.10 RISING

เป็นคำสั่งกำหนดให้เกิดการอินเตอร์รัปต์ขึ้น เมื่อเกิดการเปลี่ยนแปลงลอจิกต่ำมาเป็นลอจิกสูง ที่ขาอินพุตอินเตอร์รัปต์ภายนอก

**ตัวอย่างที่ 1-62**

```
attachInterrupt(0, myFunction, RISING);
// ไปทำงานที่ myfunction เมื่อเกิดอินเตอร์รัปต์จากการเปลี่ยนลอจิกต่ำเป็นสูง
// ที่ขาอินพุตอินเตอร์รัปต์ภายนอก 0
```

### 1.13.11 LSBFIRST

เป็นคำสั่งกำหนดลำดับของการส่งหรือเลื่อนบิตข้อมูล โดยกำหนดให้ส่งบิตนัยสำคัญต่ำสุด หรือ LSB ก่อน มักใช้ร่วมกับคำสั่ง shiftOut

### 1.13.12 MSBFIRST

เป็นคำสั่งกำหนดลำดับของการส่งหรือเลื่อนบิตข้อมูล โดยกำหนดให้ส่งบิตนัยสูงสุดหรือ MSB ก่อน มักใช้ร่วมกับคำสั่ง shiftOut

## 1.14 พอยน์เตอร์ (Pointer) หรือตัวชี้ข้อมูล

**พอยน์เตอร์ (pointer) คือ ข้อมูลหรือตัวแปรชนิดหนึ่งที่เก็บค่าเพื่อชี้ตำแหน่งของหน่วยความจำ** จะต้องกำหนดชนิดของพอยน์เตอร์ทุกครั้งที่ทำการประกาศเช่นเดียวกับข้อมูลทั่วไป ซึ่งชนิดข้อมูลของพอยน์เตอร์ต้องเป็นชนิดเดียวกับข้อมูลที่ใช้เก็บค่าซึ่งใช้ในการชี้ตำแหน่งของหน่วยความจำ

### 1.14.1 การประกาศตัวแปร pointer

คล้ายกับการประกาศตัวแปรทั่วไป ต่างกันที่ชนิดของ pointer จะต้องมีเครื่องหมาย * ต่อท้ายทุกครั้ง เรามักจะประกาศตัวแปร pointer ไว้ด้านบนสุดของฟังก์ชั่นหรือตอนต้นของโปรแกรมเช่นเดียวกับตัวแปรทั่วไป และตัวแปร pointer ที่ถูกประกาศในฟังก์ชั่นหนึ่งจะไม่สามารถเรียกใช้ได้ในอีกฟังก์ชั่นหนึ่งเช่นกัน

**รูปแบบ** :

```
datatype* var;
```

**โดยที่** datatype คือ ชนิดข้อมูลที่เก็บค่าตำแหน่งของหน่วยความจำ
var คือ ชื่อของตัวแปร pointer

## 1.14. 2 การอ่านค่าจากตัวแปร pointer

การอ่านค่าจากตัวแปร pointer ต้องทำทีละตำแหน่ง ยกเว้นการอ่านค่าข้อมูล pointer บางชนิด เช่น char สามารถใช้ได้หลายตำแหน่งพร้อมกัน

**รูปแบบ** :

var[index]

โดยที่ var คือ ชื่อของตัวแปรpointer

index คือ ตำแหน่งของตัวแปรในชุดตัวแปร pointer ที่ต้องการอ่านข้อมูล

## 1.14.3 การกำหนดค่าให้ pointer

การกำหนดค่าให้ตัวแปร pointer ทำได้ 2 แบบคือ การกำหนดค่าแอดเดรสให้ตัวแปร pointer และการกำหนดค่าให้กับหน่วยความจำในตำแหน่งที่ pointer ชี้

(1) การกำหนดค่าแอดเดรสให้ตัวแปร pointer

**รูปแบบ** :

var = value;

หรือ

var = &variable;

โดยที่ var คือ ชื่อของตัวแปร pointer

value คือ ค่าแอดเดรสหรือตำแหน่งของหน่วยความจำที่ต้องการให้ตัวแปร pointer ชี้

variable คือ ค่าแอดเดรสของหน่วยความจำของตัวแปร variable ที่ต้องการให้ตัวแปร pointer ชี้ เมื่อนำเครื่องหมาย & มาไว้หน้าตัวแปร จะเป็นการกำหนดหรือเลือกกำหนดค่าแอดเดรสของตัวแปรนั้น

(2) กำหนดค่าให้กับหน่วยความจำในตำแหน่งที่ pointer ชี้

ทำได้โดยนำเครื่องหมาย * ไว้หน้าชื่อของตัวแปร pointer ก่อน เพื่อบอกว่า ค่าที่จะกำหนดเป็นข้อมูลที่ต้องนำไปเก็บไว้ที่ตำแหน่งดังกล่าว โดยใช้เครื่องหมาย = เพื่อกำหนดค่าเช่นเดียวกับตัวแปรทั่วไป และจะกำหนดค่าให้ตัวแปรได้ก็ต่อเมื่อชนิดข้อมูลและชนิดของตัวแปร pointer เป็นชนิดเดียวกัน เช่น ถ้าข้อมูลเป็น interger ตัวแปร pointer ก็ต้องเป็นชนิด interger ด้วย

**รูปแบบ** :

*var = value;

โดยที่ var คือ ชื่อของตัวแปร pointer

value คือ ข้อมูลที่จะเก็บลงในแอดเดรสของหน่วยความจำที่ pointer ชี้

## 1.15 โอเวอร์โหลด (Overload)

เป็นความสามารถที่มีอยู่ใน C++ โดยการโอเวอร์โหลดคือ **การประกาศชื่อฟังก์ชั่นหลายๆ ฟังก์ชั่นเป็นชื่อเดียวกันได้ โดยมีพารามิเตอร์ที่แตกต่างกันอาจเป็นขนาดหรือชนิดของข้อมูล แต่ทุกฟังก์ชั่นต้องคืนค่าเป็นข้อมูลชนิดเดียวกัน** โดยโปรแกรมจะทราบว่าเรากำลังเรียกใช้ฟังก์ชั่นใดได้จากพารามิเตอร์ขณะเรียกใช้งาน

บางครั้งฟังก์ชั่นที่มีการโอเวอร์โหลด อาจทำให้เกิดความผิดพลาดเนื่องจากความไม่ชัดเจนของชนิดข้อมูลที่เป็นพารามิเตอร์ทำให้โปรแกรมไม่สามารถตัดสินใจได้ว่าควรเรียกใช้ฟังก์ชั่นใด

## 1.16 โปรแกรมเชิงวัตถุ (Object-Oriented Programming : OOP)

สิ่งที่ทำให้ที่ภาษา C++ แตกต่างจากภาษา C ก็คือ ภาษา C++ สามารถเขียนโปรแกรมเชิงวัตถุหรือ Object-Oriented Programming (OOP) ได้ ดังนั้นภาษา C++ จึงมีคลาส (Class) และความสามารถต่างๆ ของคลาส เช่น การสืบทอด (Inheritance) และการแก้ไข (override) ซึ่งเป็นสิ่งสำคัญที่ทำให้ภาษา C++ สามารถเขียนโปรแกรมแบบ OOP ได้ ดังนั้น การเขียนโปรแกรมใน Wiring จึงสามารถรองรับการเขียนโปรแกรมในลักษณะ OOP ได้

ในที่นี้ขออธิบายข้อมูลเบื้องต้นที่ควรทราบเกี่ยวกับโปรแกรมเชิงวัตถุหรือ OOP เพื่อใช้เป็นข้อมูลอ้างอิงในขั้นต้นสำหรับการเขียนโปรแกรม OOP ใน Wiring

### 1.16.1 คลาส (Class)

คลาสในภาษา C++ คือขอบเขต (encapsulation) ที่ครอบคลุมข้อมูลและฟังก์ชั่นที่ใช้ในการจัดการข้อมูลนั้น

#### 1.16.1.1 ข้อมูลและตัวแปรในคลาส

ข้อมูลที่เป็นสมาชิกของคลาสสามารถเป็นชนิดใดก็ได้และสามารถประกาศเป็นพอยน์เตอร์ได้อีกด้วย คลาสจะเป็นเหมือนตัวแปรตัวหนึ่งหรือที่เราเรียกว่า ออบเจ็กต์ (object)

### 1.16.1.2 ฟังก์ชั่นในคลาส

ฟังก์ชั่นที่สามารถประกาศภายในคลาสได้มี 3 ชนิดคือ

1. ฟังก์ชั่นทั่วไป

2. Constructor เป็นฟังก์ชั่นที่ใช้กำหนดค่าเริ่มต้นให้กับออบเจ็กต์ เป็นฟังก์ชั่นพิเศษที่จะมีชื่อเดียวกับชื่อคลาสสามารถโอเวอร์โหลดได้ไม่จำกัดภายในคลาส ซึ่งแบ่งตามพารามิเตอร์ที่ต่างกัน Constructor จะไม่มีการคืนค่าใดๆ การสร้างฟังก์ชั่นแบบ Constructor มีข้อกำหนดดังนี้

2.1 ต้องมีชื่อเดียวกับคลาส

2.2 ต้องไม่มีการคืนค่าใดๆ

2.3 constructor จะถูกเรียกขึ้นมาเป็นอันดับแรกเมื่อมีการสร้างออบเจ็กต์ของคลาส การกำหนดค่าเริ่มต้นต่างๆ เช่น การประกาศสมาชิกในคลาสและการจองพื้นที่หน่วยความจำ

3. Destructor คือฟังก์ชั่นที่ใช้ในการทำลายออบเจ็กต์ เป็นฟังก์ชั่นพิเศษเช่นเดียวกับ Constructor ซึ่งมีชื่อเดียวกับชื่อคลาส แต่จะขึ้นต้นด้วยเครื่องหมาย ~ โดย Destructor จะไม่มีการคืนค่าใดๆ และจะถูกเรียกเมื่อออบเจ็กต์ของคลาสนั้นๆ สิ้นสุดการทำงาน

## 1.16.2 ระดับการเข้าถึงข้อมูลและฟังก์ชั่น

คลาสใน C++ มีการกำหนดระดับการเข้าถึงที่สำคัญอยู่ 3 ระดับคือ Private, Public และ Protected ดังนี้

1. private คือ การกำหนดให้เข้าถึงข้อมูลได้โดยฟังก์ชั่นที่เป็นสมาชิกของคลาสนั้นๆ

2. protected คือ การกำหนดให้เข้าถึงข้อมูลได้โดยฟังก์ชั่นที่เป็นสมาชิกของคลาสและคลาสที่สืบทอดจากคลาสนั้นๆ เท่านั้น

3. public คือ การกำหนดให้เข้าถึงได้โดยสมาชิกอื่นๆ จากภายนอก โดยทั่วไปไม่นิยมประกาศข้อมูลไว้ในส่วนนี้

## 1.16.3 การประกาศคลาส

เราจะเรียกใช้งานข้อมูลได้ก็ต่อเมื่อมีตัวแปรมาเก็บข้อมูลและต้องมีการประกาศตัวแปรเสียก่อน เรามักจะประกาศตัวแปรไว้ด้านบนสุดของฟังก์ชั่นหรือตอนต้นโปรแกรม ตัวแปรที่ถูกประกาศในฟังก์ชั่นหนึ่งจะไม่สามารถเรียกใช้ได้ในฟังก์ชั่นอื่น ตัวแปรชนิดเดียวกันสามารถประกาศพร้อมกันหลายตัวได้ โดยใช้เครื่องหมาย , เป็นตัวแยก

**รูปแบบ** :

```
class name{
    private:
        statement;      // การประกาศตัวแปรและฟังก์ชั่นต่างๆ ที่เป็น private
    protected:
        statement;      // การประกาศตัวแปรและฟังก์ชั่นต่างๆ ที่เป็น protected
    public:
        statement;      // การประกาศตัวแปรและฟังก์ชั่นต่างๆ ที่เป็น public
};
```

โดยที่ name คือ ชื่อของคลาส ซึ่งมีกฎในการตั้งชื่อเช่นเดียวกับชื่อตัวแปร

statement คือ คำสั่งหรือฟังก์ชั่นต่างๆ อาจมีได้หลายคำสั่ง

**<u>ตัวอย่างที่ 1-63</u>**

```
class Example_class            // Sample Class for the C++ Tutorial
{
    private:
        int x;                 // Data member
        int y;                 // Data member
    public:
        Example_Class()        // Constructor for the C++ tutorial
        {
            x = 0;
            y = 0;
        }
        ~Example_Class()       // Destructor for the C++ Tutorial
        { }
    int Add()
    {
        return x+y;
    }
};
```

### 1.16.4 friend

ข้อมูลต่างๆ ที่ประกาศเป็น private ภายในคลาสจะไม่สามารถเข้าถึงได้จากฟังก์ชั่นที่ไม่ใช่สมาชิกของคลาสหรือคลาสภายนอก แต่ใน C++ มีข้อยกเว้นที่จะยอมให้ฟังก์ชั่นและคลาสภายนอกเข้าถึงข้อมูลที่ประกาศเป็น private ได้ โดยประกาศให้ฟังก์ชั่นและคลาสภายนอกนั้นเป็น friend ภายในคลาส

**รูปแบบ** :

friend class name; หรือ friend type function(parameters);

### 1.16.5 การสืบทอด (inheritance)

**การสืบทอดคือการสร้างคลาสใหม่หรือคลาสลูก (derived class) ที่สืบทอดจากคลาสที่มีอยู่แล้วหรือจากคลาสแม่ (base class)** โดยคลาสลูกที่สร้างใหม่จะได้รับสืบทอดตัวแปรและฟังก์ชั่นของคลาสแม่มาด้วย ยกเว้น constructor, destructor, friend ฟังก์ชั่นและ friend คลาส นอกจากนั้น คลาสลูกยังเพิ่มตัวแปรและฟังก์ชั่นของตัวเองได้ และแก้ไขหรือโอเวอร์ไรด์ (override) บางฟังก์ชั่นของคลาสแม่ได้ด้วย (การแก้ไขหรือ override เป็นการแก้ไขฟังก์ชั่นต่างๆ ที่ได้รับการสืบทอดมาจากคลาสแม่ โดยฟังก์ชั่นดังกล่าวจะต้องมีชื่อและpพารามิเตอร์เหมือนเดิม) การสืบทอดจะช่วยลดเวลาในการเขียนโค้ดที่ซ้ำซ้อนกัน

**รูปแบบ** :

```
class derivedClass : public baseClass {
   private:
      statement;      // การประกาศตัวแปรและฟังก์ชั่นต่าง ๆ ที่เป็น private
   protected:
      statement;      // การประกาศตัวแปรและฟังก์ชั่นต่าง ๆ ที่เป็น protected
   public:
      statement;      // การประกาศตัวแปรและฟังก์ชั่นต่าง ๆ ที่เป็น public
};
```

โดยที่ derivedClass คือ ชื่อของคลาสลูกที่ต้องการสร้างใหม่ ซึ่งมีกฎในการตั้งชื่อเช่นเดียวกับชื่อตัวแปร
baseClass คือ ชื่อของคลาสแม่ที่ต้องการสืบทอด
statement คือ คำสั่งหรือฟังก์ชั่นต่าง ๆ อาจมีได้หลายคำสั่ง

คลาสลูกไม่สามารถเข้าถึงสมาชิกของคลาสแม่ที่กำหนดเป็น private ได้ แต่เข้าถึงสมาชิกของคลาสแม่ที่กำหนดเป็น protected ได้

# บทที่ 2
# ฟังก์ชั่นโปรแกรมภาษา C/C++ ใน Wiring

นอกจากคำสั่งพื้นฐานที่มีให้ใช้ในภาษา C/C++ แล้ว ใน Wiring ยังมีฟังก์ชั่นและคำสั่งพิเศษเพิ่มเติมให้ใช้ที่จะช่วยให้เขียนโปรแกรมได้ง่ายขึ้น เช่น ฟังก์ชั่นในการแปลงข้อมูล, ฟังก์ชั่นเกี่ยวกับตรีโกณมิติ เป็นต้น

## 2.1 ฟังก์ชั่นในการแปลงข้อมูล (Conversion)

ใน Wiring มีฟังก์ชั่นที่ใช้ในการแปลงข้อมูลอยู่ ซึ่งสามารถเรียกใช้ได้เลยโดยไม่ต้องผนวกหรือนำเข้ามาจากภายนอก โดยมีรายละเอียดดังนี้

### 2.1.1 boolean()

ใช้ในการแปลงข้อมูล (value) ให้เป็นชนิด boolean

**รูปแบบ** :

boolean(value);

โดยที่ value คือ ข้อมูลชนิด int, long, float, char, byte หรือ boolean

**ตัวอย่างที่ 2-1**

```
boolean b = boolean(60);     // กำหนดให้ b มีค่าเป็นจริง (true)
boolean a = boolean(0);      // กำหนดให้ a มีค่าเป็นเท็จ (false)
```

### 2.1.2 byte()

ใช้ในการแปลงข้อมูล (value) ให้เป็นชนิด byte

**รูปแบบ** :

byte(value);

โดยที่ value คือ ข้อมูลชนิด int, long, float, char, byte หรือ boolean

**ตัวอย่างที่ 2-2**

```
byte b = byte(260);
```

## 2.1.3 char()

ใช้ในการแปลงข้อมูลให้เป็นชนิด char

**รูปแบบ :**

```
char(value);
```

โดยที่ value คือ ข้อมูลชนิด int, long, float, char, byte หรือ boolean

**ตัวอย่างที่ 2-3**

```
char c = char(126);
```

## 2.1.4 int()

ใช้ในการแปลงข้อมูลให้เป็นชนิด integer หรือตัวเลขที่ไม่มีจุดทศนิยม

**รูปแบบ :**

```
int(value);
```

โดยที่ value คือ ข้อมูลชนิด int, long, float, char, byte หรือ boolean

**ตัวอย่างที่ 2-4**

```
int i = int(245.456);
```

## 2.1.5 long()

ใช้ในการแปลงข้อมูลให้เป็นชนิด long (ไม่มีจุดทศนิยม)

**รูปแบบ :**

```
long(value);
```

โดยที่ value คือ ข้อมูลชนิด integer, long, float, char, byte หรือ boolean

**ตัวอย่างที่ 2-5**

```
long i = long(245.456);
```

## 2.1.6 float()

ใช้ในการแปลงข้อมูลให้เป็นชนิด float

**รูปแบบ :**

```
float(value);
```

โดยที่ value คือ ข้อมูลชนิด integer, long, float, char, byte หรือ boolean

**ตัวอย่างที่ 2-6**

```
float f = float(true);
```

## 2.2 ฟังก์ชั่นในการคำนวณ (Calculation)

ใน Wiring มีฟังก์ชั่นที่ใช้ในการคำนวณ เช่น การหาค่ามากที่สุด, การหาค่าน้อยที่สุด เป็นต้น ซึ่งช่วยให้เขียนโปรแกรมได้ง่ายขึ้น คำสั่งเหล่านี้สามารถเรียกใช้ได้โดยไม่ต้องผนวกไฟล์ไลบรารีจากภายนอก โดยมีรายละเอียดดังนี้

### 2.2.1 min()

เป็นคำสั่งหาค่าน้อยที่สุดในชุดข้อมูล ฟังก์ชั่นนี้จะคืนค่าเป็นค่าที่น้อยที่สุดของชุดข้อมูลที่นำมาตรวจสอบ สามารถใช้กับข้อมูลชนิด int หรือ float ขึ้นอยู่กับชนิดของข้อมูลอินพุต

**รูปแบบ** :

min(value1, value2);

หรือ

min(value1, value2, value3);

โดยที่ value คือ ข้อมูลหรือตัวแปรที่ต้องการหาค่าที่น้อยที่สุดซึ่งเป็นตัวเลขชนิด int หรือ float

**<u>ตัวอย่างที่ 2-7</u>**

```
int d = min(5, 9);                  // ผลลัพธ์ที่ได้คือ d = 5
int e = min(-4, -12);               // ผลลัพธ์ที่ได้คือ e = -12
float f = min(12.3, 230.24);        // ผลลัพธ์ที่ได้คือ f = 12.3
```

### 2.2.2 max()

เป็นคำสั่งหาค่าที่มากที่สุดในชุดข้อมูล ฟังก์ชั่นนี้จะคืนค่าเป็นค่าที่มากที่สุดของชุดข้อมูลที่นำมาตรวจสอบ สามารถใช้กับข้อมูลชนิด int หรือ float ขึ้นอยู่กับชนิดของข้อมูลอินพุต

**รูปแบบ** :

max(value1, value2);

หรือ

max(value1, value2, value3);

โดยที่ value คือ ข้อมูลหรือตัวแปรที่ต้องการหาค่าที่มากที่สุด ซึ่งเป็นตัวเลขชนิด int หรือ float

**<u>ตัวอย่างที่ 2-8</u>**

```
int a = max(5, 9);                  // ผลลัพธ์ที่ได้คือ a = 9
int b = max(-4, -12);               // ผลลัพธ์ที่ได้คือ b = -4
float c = max(12.3, 230.24);        // ผลลัพธ์ที่ได้คือ c = 230.24
```

## 2.2.3 abs()

เป็นคำสั่งคำนวณหาค่าสัมบูรณ์ (absolute) หรือปริมาณ (magnitude) ของข้อมูลตัวเลข ซึ่งผลลัพธ์ที่ได้มีค่าเป็นค่าบวกเสมอ ฟังก์ชั่นนี้คืนค่าเป็นค่าสัมบูรณ์ ซึ่งจะเป็นข้อมูลชนิด int หรือ float ขึ้นอยู่กับชนิดของข้อมูลอินพุต

**รูปแบบ** :

abs(value);

โดยที่ value คือ ข้อมูลหรือตัวแปรที่ต้องการหาค่าสมบูรณ์ซึ่งเป็นตัวเลขชนิด int หรือ float

**ตัวอย่างที่ 2-9**

```
int a = abs(153);          // กำหนดให้ a มีค่าเป็น 153
int b = abs(-15);          // กำหนดให้ b มีค่าเป็น 15
float c = abs(12.234);     // กำหนดให้ c มีค่าเป็น 12.234
float d = abs(-9.23);      // กำหนดให้ d มีค่าเป็น 9.23
```

## 2.2.4 constrain()

เป็นคำสั่งจำกัดค่าของข้อมูลให้อยู่ในขอบเขตของค่าสูงสุด (max) และค่าต่ำสุด (min)

(1) ถ้าค่าของข้อมูล (value) มากกว่าค่าสูงสุด ฟังก์ชั่น constrain() จะคืนค่าเป็นค่าที่มากที่สุด

(2) ถ้าหากค่าของข้อมูล (value) น้อยกว่าค่าต่ำสุด ฟังก์ชั่น constrain() จะคืนค่าเป็นค่าที่น้อยที่สุด (min)

(3) ถ้าค่าของข้อมูล (value) อยู่ในช่วงที่กำหนด ฟังก์ชั่น constrain() จะคืนค่าเป็นค่าเดิม

ชนิดของข้อมูลที่คืนอาจเป็นชนิด int หรือ float ขึ้นอยู่กับข้อมูลอินพุต

**รูปแบบ** :

constrain (value, min, max);

โดยที่ value คือ ข้อมูลหรือตัวแปรที่ต้องการจำกัดขอบเขต ซึ่งเป็นตัวเลขชนิด int หรือ float
min คือ ค่าต่ำสุดของขอบเขตข้อมูล ซึ่งเป็นตัวเลขชนิด int หรือ float
max คือ ค่าสูงสุดของขอบเขตข้อมูล ซึ่งเป็นตัวเลขชนิด int หรือ float

**ตัวอย่างที่ 2-10**

```
int data = 100;
int t = constrain(data, 30, 70);    // ค่าของ t จะถูกเปลี่ยนเป็น 70 ซึ่งอยู่ในช่วง 30 ถึง 70
```

## 2.2.5 round()

เป็นคำสั่งที่ใช้คำนวณหาค่าเลขจำนวนเต็มที่ใกล้เคียงกับค่าของข้อมูล (value) มากที่สุด เช่น round(9.1); จะได้ผลลัพธ์เป็น 9 คืนค่าเป็นข้อมูลชนิด int เสมอ

**รูปแบบ** :

round (value);

โดยที่ value คือ ข้อมูลหรือตัวแปรชนิด float

**ตัวอย่างที่ 2-11**

```
float x = 8.22;
int rx = round(x);         // ผลลัพธ์ที่ได้คือ rx = 8
float y = 9.9;
int ry = round(y);         // ผลลัพธ์ที่ได้คือ ry = 10
float z = -10.6;
int rz = round(z);         // ผลลัพธ์ที่ได้คือ rz = -11
```

## 2.2.6 floor()

เป็นคำสั่งคำนวณหาค่าที่ใกล้เคียงกับค่าของข้อมูล (value) มากที่สุด โดยค่าที่ได้จะต้องน้อยกว่าหรือเท่ากับค่าที่เป็นพารามิเตอร์ คืนค่าเป็นข้อมูลชนิด float

**รูปแบบ** :

floor (value);

โดยที่ value คือ ข้อมูลหรือตัวแปรชนิด float

**ตัวอย่างที่ 2-12**

```
float x = 2.88;
float a = floor(x);        // ผลลัพธ์ที่ได้คือ a = 2.0
```

## 2.2.7 ceil()

เป็นคำสั่งคำนวณหาค่าที่ใกล้เคียงกับค่าของข้อมูล (value) มากที่สุด โดยค่าที่ได้จะต้องมากกว่าหรือเท่ากับค่าที่เป็นพารามิเตอร์ คืนค่าเป็นข้อมูลชนิด float

**รูปแบบ** :

ceil (value);

โดยที่ value คือ ข้อมูลหรือตัวแปรชนิด float

**ตัวอย่างที่ 2-13**

```
float x = 8.22;
float a = ceil(x);         // ผลลัพธ์ที่ได้คือ a = 9.0
```

## 2.2.8 pow()

เป็นคำสั่งคำนวณหาค่าเลขยกกำลัง คืนค่าเป็นผลลัพธ์ของเลขยกกำลังซึ่งเป็นข้อมูลชนิด float

**รูปแบบ :**

pow(base, exponent);

โดยที่ base คือ เลขฐาน เป็นตัวเลขชนิด int หรือ float

exponent คือ เลขชี้กำลัง เป็นตัวเลขชนิด int หรือ float

**ตัวอย่างที่ 2-14**

```
float a = pow( 1, 3);        // a มีค่าเป็น 13 หรือ 1*1*1 = 1
float b = pow( 3, 5);        //  b มีค่าเป็น 35หรือ 3*3*3*3*3 = 243
float c = pow( 3,-5);        //  c มีค่าเป็น 1 / 3*3*3*3*3 = 1 / 243 = .0041
float d = pow(-3, 5);        // d มีค่าเป็น -3*-3*-3*-3*-3 = -243
```

## 2.2.9 sq()

เป็นคำสั่งคำนวณหาค่าของเลขยกกำลัง 2 คืนค่าผลลัพธ์เป็นบวก และค่าที่ได้เป็นข้อมูลชนิด float

**รูปแบบ :**

sq(value);

โดยที่ value คือ ข้อมูลหรือตัวแปรชนิด int หรือ float

**ตัวอย่างที่ 2-15**

```
float a = sq(1);         //  a มีค่าเป็น 1 มาจาก 1 x 1=1
float b = sq(-5);        //  b มีค่าเป็น 25 มาจาก 5 x 5 = 25
float c = sq(9);         //  c มีค่าเป็น 81 มาจาก 9 x 9=81
```

## 2.2.10 sqrt()

เป็นคำสั่งที่ใช้ในการคำนวณหาค่าของรากที่สอง (square root) ของตัวเลข คืนค่าเป็นผลลัพธ์ของการถอดรากที่สองของค่านั้น ฟังก์ชั่น sqrt() คืนค่าเป็นบวกเสมอ และค่าที่คืนจะเป็นข้อมูลชนิด float

**รูปแบบ :**

sqrt(value);

โดยที่ value คือ ข้อมูลหรือตัวแปรชนิด int หรือ float ซึ่งเป็นเลขบวกเสมอ

**ตัวอย่างที่ 2-16**

```
float a = sqrt(6561);      // a มีค่าเป็น 81 มาจาก 81 x 81 = 6561
float b = sqrt(625);       // b มีค่าเป็น 25 มาจาก 25 x 25 =625
float c = sqrt(1);         // c มีค่าเป็น 1  มาจาก 1 x 1 = 1
```

## 2.2.11 map()

เป็นคำสั่งแปลงค่าของข้อมูลให้อยู่ในช่วงที่ต้องการ ค่าที่คืนจากฟังก์ชั่นเป็นข้อมูลชนิด float

**รูปแบบ :**

map(value, low1, high1, low2, high2);

**โดยที่** value คือ ข้อมูลหรือตัวแปรที่ต้องการแปลงค่า เป็นข้อมูลชนิด float

low1 คือ ขอบเขตล่างเดิมของข้อมูลหรือตัวแปรที่ต้องการแปลงค่าเป็นข้อมูลชนิด float

high1 คือ ขอบเขตบนเดิมของข้อมูลหรือตัวแปรที่ต้องการแปลงค่า เป็นข้อมูลชนิด float

low2 คือ ขอบเขตล่างใหม่ของข้อมูลหรือตัวแปรที่ต้องการแปลงค่าเป็นข้อมูลชนิด float

high2 คือ ขอบเขตบนใหม่ของข้อมูลหรือตัวแปรที่ต้องการแปลงค่า เป็นข้อมูลชนิดfloat

**ตัวอย่างที่ 2-17**

```
int value;
int m;
void setup()
{
    Serial.begin(9600);
}
void loop()
{
    value = analogRead(0);                  // จากเดิมค่า m อยู่ในช่วง 0 ถึง 1023
    m = (int)map(value, 0, 1023, 0, 255);   // ใช้map() เพื่อแปลงให้อยู่ในช่วง  0 ถึง 255
    Serial.println(m);                      // แสดงผลค่าของตัวแปร m
    delay(100);
}
```

# 2.3 ฟังก์ชั่นตรีโกณมิติ (Trigonometry)

ใน Wiring มีฟังก์ชั่นที่ใช้ในการคำนวณค่ามุมและตรีโกณมิติ ดังต่อไปนี้

## 2.3.1 sin()

เป็นคำสั่งคำนวณหาค่าของไซน์ (sine) ของมุม โดยค่าของมุมต้องอยู่ในหน่วยเรเดียน ซึ่งมีค่าระหว่าง 0 ถึง 6.28 และจะคืนค่าอยู่ในช่วง -1 ถึง 1 เป็นข้อมูลชนิด float

**รูปแบบ** :

```
sin(angle);
```

โดยที่ angle คือ มุมที่ต้องการคำนวณอยู่ในหน่วยเรเดียน (PI ถึง -PI) และต้องเป็นข้อมูลชนิด float

## 2.3.2 cos()

เป็นคำสั่งคำนวณหาค่าของโคไซน์ (cosine) ของมุม โดยค่าของมุมต้องอยู่ในหน่วยเรเดียน ซึ่งมีค่าระหว่าง 0 ถึง 6.28 และจะคืนค่าอยู่ในช่วง -1 ถึง 1 เป็นข้อมูลชนิด float

**รูปแบบ** :

```
cos(angle);
```

โดยที่ angle คือ มุมที่ต้องการคำนวณอยู่ในหน่วยเรเดียน (PI ถึง -PI) และต้องเป็นข้อมูลชนิด float

## 2.3.3 tan()

เป็นคำสั่งหาค่าของแทนเจนต์ (tangent : tan) ของมุม ซึ่งเป็นอัตราส่วนของ sine และ cosine ค่าของมุมอยู่ในหน่วยเรเดียน มีค่า 0 ถึง 6.28 และจะคืนค่าอยู่ในช่วง - infinity ถึง infinity เป็นข้อมูลชนิด float

**รูปแบบ** :

```
tan(angle);
```

โดยที่ angle คือ มุมที่ต้องการคำนวณอยู่ในหน่วยเรเดียน (PI ถึง -PI) และต้องเป็นข้อมูลชนิด float

## 2.3.4 atan2()

เป็นคำสั่งคำนวณหาค่ามุมในหน่วยเรเดียน ซึ่งวัดจากจุดพารามิเตอร์เทียบกับจุดตั้งต้นหรือ origin โดยวัดจากแกน x ทางบวก ฟังก์ชั่นจะคืนค่าในช่วง PI ถึง -PI เป็นชนิด float ฟังก์ชั่น atan2() นิยมใช้ในการหาตำแหน่งของเคอร์เซอร์

**รูปแบบ** :

```
atan2(y, x);
```

โดยที่ y คือ จุดในแกน y เป็นข้อมูลชนิด int หรือ float

x คือ จุดในแกน x เป็นข้อมูลชนิด int หรือ float

การกำหนดค่าพารามิเตอร์ของคำสั่ง atan2() ค่าของจุดในแกน y ต้องมาก่อนจุดในแกน x เสมอ

## 2.3.5 degrees()

เป็นคำสั่งที่ใช้ในการแปลงมุมในหน่วยเรเดียนให้เป็นมุมในหน่วยองศา (degree) โดย 2Pi เรเดียน = 360 องศา ค่าที่คืนมาเป็นตัวเลขชนิด float

**รูปแบบ** :

```
degrees(angle);
```

โดยที่ angle คือมุมที่ต้องการคำนวณในหน่วยเรเดียน (PI ถึง -PI) และต้องเป็นข้อมูลชนิด int หรือ float

**ตัวอย่างที่ 2-18**

```
void setup()
{
    Serial.begin(9600);          // ตั้งค่าอัตราบอดให้กับพอร์ตอนุกรมเป็น 9600 บิตต่อวินาที
}
void loop()
{
    float rad = PI/4;            // กำหนดค่าให้ตัวแปร rad เป็นมุมแบบ radian มีค่า PI/4
    float deg = degrees(rad);    // แปลงมุมในหน่วยเรเดียนให้เป็นองศา Pi/4 เท่ากับ 45.0 องศา
    Serial.print(rad);           // แสดงค่าของตัวแปร rad
    Serial.print(" radians is ");
    Serial.print(deg);           // แสดงค่าของตัวแปร deg
    Serial.println(" degrees");
}
```

## 2.3.6 radians()

เป็นคำสั่งแปลงมุมในหน่วยองศา (degree) ให้เป็นมุมในหน่วยเรเดียน โดย 360 องศา = 2PI = 3.1428 เรเดียน เช่น 90 องศา = PI/2 = 1.5707964 ค่าที่คืนมาเป็นตัวเลขชนิด float

**รูปแบบ** :

```
radians(angle);
```

โดยที่ angle คือมุมที่ต้องการคำนวณในหน่วยเรเดียน (PI ถึง -PI) และต้องเป็นข้อมูลชนิดinteger หรือ float

**ตัวอย่างที่ 2-19**

```
float deg = 45.0;
float rad = radians(deg);
```

## 2.4 ฟังก์ชั่นสุ่มค่า

ใน Wiring มีฟังก์ชั่นที่ใช้ในการสุ่มตัวเลขหรือ Random มีรายละเอียดดังนี้

### 2.4.1 random()

เป็นคำสั่งสุ่มตัวเลข ในแต่ละครั้งที่เรียกใช้ฟังก์ชั่น random() จะมีการคืนค่าเป็นตัวเลขชนิด float หรือตัวเลขที่มีจุดทศนิยมที่อยู่ในช่วงที่กำหนดไว้ (value) หากในการเรียกใช้ฟังก์ชั่นกำหนดค่าให้เพียง 1 ค่า โปรแกรมทำการสุ่มตัวเลขในช่วงตั้งแต่ 0 ถึงค่าที่กำหนดไว้ ฟังก์ชั่น random() สามารถเรียกใช้ได้ 2 รูปแบบดังนี้

**รูปแบบ** :

random(value1);

หรือ

random(value1, value2);

โดยที่ value คือข้อมูลชนิด int หรือ float

**<u>ตัวอย่างที่ 2-20</u>**

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

void loop()
{
    int val = random(-40, 40);    // กำหนดให้ val เป็นตัวแปรเก็บค่าที่ได้จากการสุ่มในช่วง -40 ถึง 40
    Serial.print("val = ");
    Serial.println(val);
    int x = random(5);            // กำหนดให้ x เป็นตัวแปรเก็บค่าที่ได้จากการสุ่มในช่วง 0 ถึง 5
    Serial.print("x = ");
    Serial.println(x);
}
```

## 2.4.2 randomSeed()

เป็นคำสั่งกำหนดค่าเริ่มต้นให้กับคำสั่ง random() เพื่อกำหนดผลที่ได้จากการสุ่มค่า

**รูปแบบ :**

randomSeed(value);

**โดยที่** value คือ ข้อมูลชนิด int

**ตัวอย่างที่ 2-21**

```
void setup()
{
    Serial.begin(9600);
    randomSeed(20);                 // กำหนดค่าเริ่มต้นให้การสุ่ม
}
void loop()
{
    int randNum = random(0, 500);
    Serial.println(randNum, 2);
    delay(500);
}
```

คำสั่ง random() จะทำการสุ่มค่าให้ผลเป็นเลขชุดหนึ่ง แม้เมื่อเริ่มการทำงานใหม่ก็จะสุ่มได้ตัวเลขชุดเดิม ในกรณีที่ค่าพารามิเตอร์ต่างๆเหมือนกัน

คำสั่ง randomSeed() จะทำให้ผลของการสุ่มเปลี่ยนไปแม้ว่าค่าของพารามิเตอร์ต่างๆ ของคำสั่ง random() จะเหมือนกัน ดังนั้นหากมีความต้องการสุ่มค่าแบบมีความกระจายของข้อมูลสูงควรใช้คำสั่ง randomSeed(); เพื่อกำหนดค่าเริ่มต้น โดยค่า value ควรนำมาจากแหล่งกำเนิดข้อมูลที่มีการเปลี่ยนแปลงตลอดเวลา เช่น ค่าจากฐานเวลา หรือค่าจากการแปลงสัญญาณอะนาลอกเป็นดิจิตอล จากนั้นจึงใช้คำสั่ง random() เพื่อสุ่มค่าต่อไป

## 2.5 ฟังก์ชั่นเกี่ยวกับเวลา

ใน Wiring มีฟังก์ชั่นที่ใช้ในกำหนดค่าหน่วงเวลา (delay) และคำนวณเวลา ซึ่งสามารถเรียกใช้ได้โดยไม่ต้องผนวกไฟล์ไลบรารีเพิ่มเติม มีรายละเอียดดังนี้

### 2.5.1 delay()

เป็นคำสั่งที่ทำให้เกิดการหน่วงเวลาในหน่วยมิลลิวินาที

**รูปแบบ** :

```
delay(milliseconds);
```

โดยที่ milliseconds คือ ข้อมูลหรือตัวแปรชนิด int เป็นตัวกำหนดค่าเวลาในหน่วยมิลลิวินาที โดยที่ 1 วินาที = 1000 มิลลิวินาที

### 2.5.2 delayMicroseconds()

เป็นคำสั่งที่ทำให้เกิดการหน่วงเวลาในหน่วยไมโครวินาที

**รูปแบบ** :

```
delayMicroseconds(microseconds);
```

โดยที่ microseconds คือ ข้อมูลหรือตัวแปรชนิด long เป็นตัวกำหนดค่าเวลาในหน่วย ไมโครวินาที โดยที่ 1 วินาที = 1,000,000 ไมโครวินาที

### 2.5.3 millis()

เป็นคำสั่งคืนค่าเวลาที่นับได้ตั้งแต่โปรแกรมเริ่มทำงานในหน่วยมิลลิวินาที ค่าที่คืนเป็นข้อมูลชนิด long

**รูปแบบ** :

```
millis();
```

**ตัวอย่างที่ 2-22**

ตัวอย่างนี้แสดงการใช้คำสั่ง millis ในการสร้างวงจรตั้งเวลาเพื่อควบคุมการทำงานของ LED ให้ติดสว่าง หลังจากที่เวลาผ่านไป 7 วินาที โดยจะเริ่มนับเวลาเมื่อมีการกดสวิตช์

```
int ledPin = 17;            // ต่อ ZX-LED กับขาพอร์ต 48 ของแผงวงจร IPST-SE
int switchPin = 22;         // เลือกใช้สวิตช์ SW1 ที่ต่อกับพอร์ต 22 ของแผงวงจร IPST-SE
long previousTime = 0;      // กำหนดตัวแปรเก็บค่าเวลาขณะที่กดสวิตช์
boolean press = false;      // ตัวแปรบอกสถานะการกดสวิตช์
```

```
void setup()
{
    pinMode(ledPin, OUTPUT);       // กำหนดให้ขาพอร์ต 17 ทำงานเป็นเอาต์พุต
    pinMode(switchPin, INPUT);     // กำหนดให้ขาพอร์ต 22 ทำงานเป็นอินพุต
}
void loop()
{
    if(digitalRead(switchPin) == LOW)  // ตรวจสอบสวิตช์ว่ามีการกดหรือไม่
    {
        previousTime = millis();        // เก็บค่าเวลาจริง ณ ปัจจุบันเป็นจุดเริ่มต้นนับเวลา
        press = true;                   // มีการกดสวิตช์
    }
    // ตรวจสอบว่า หลังจากกดสวิตช์เวลาผ่านไป 7 วินาทีแล้วหรือไม่
    if(((millis() - previousTime) >= 7000 )&&(press == true))
    {
        previousTime = 0;               // รีเซตหรือเคลียร์เวลาที่ถูกเก็บไว้
        digitalWrite(ledPin, HIGH);     // ขับ LED ให้สว่าง
        delay(1000);                    // หน่วงเวลาให้ LED สว่างนานประมาณ 1 วินาที
        digitalWrite(ledPin, LOW);      // ดับ LED
        press = false;                  // รีเซตหรือเคลียร์สถานะการกดสวิตช์
    }
}
```

เมื่อทำการรันโปรแกรม LED ที่ขาพอร์ต 17 จะดับ เมื่อมีการกดสวิตช์ที่ขาพอร์ต 22 (SW1) และเวลาผ่านไป 7 วินาที LED จะติดสว่าง 1 วินาที แล้วดับลงอีกครั้ง โปรแกรมจะทำการตรวจสอบค่าที่ขาพอร์ต 22 ว่าเป็นลอจิกต่ำหรือไม่ หากมีการกดสวิตช์ ค่าลอจิกที่ขาพอร์ต 22 จะเป็นลอจิกต่ำ ตัวแปร press ซึ่งเป็นตัวแปรบอกสถานะการกดสวิตช์ มีค่าเป็น true

จากนั้นตรวจสอบในขั้นต่อไปว่า หลังจากกดสวิตช์เวลาผ่านไป 7 วินาทีแล้วหรือไม่ ถ้าเงื่อนไขนี้เป็นจริง LED ถูกขับให้สว่างประมาณ 1 วินาที แต่ถ้าหากค่าที่ได้จากคำสั่ง digitalRead() มีค่าเป็น HIGH แสดงว่าไม่มีการกดสวิตช์ ก็จะไม่มีการนับเวลาเพื่อทำให้ LED ติดสว่าง และจะวนตรวจสอบเช่นนี้ไปตลอด เนื่องจากคำสั่งทั้งหมดนั้นอยู่ในฟังก์ชั่น loop()

## 2.6 String

ใน Wiring มี String class ที่ใช้สำหรับตัวอักษรหรือ string โดยเฉพาะ ซึ่งความแตกต่างระหว่าง string class และอะเรย์ของ char ก็คือ String class มีกระบวนการช่วยในการจัดการกับข้อมูลตัวอักษรเพื่อลดความยุ่งยากในการเขียนโปรแกรม เช่น ความสามารถในการพิจารณาแยกแต่ละตัวอักษร, การเปรียบเทียบ string, การค้นหา string, การแยกส่วนของ string, และการแปลง string ให้เป็นตัวพิมพ์เล็ก (lower case) และตัวพิมพ์ใหญ่(upper case)

Wiring String class มีความแตกต่างกับ String class ของ Java ตรงที่บางกระบวนการหรือเมธอดจะแก้ไขค่าของ string นั้นจริงๆ ไม่ใช่เพียงคืนค่าที่เป็นผลจากการทำเมธอดนั้นๆ

***ข้อมูลชนิด String จะอยู่ในเครื่องหมายอัญประกาศหรือเครื่องหมายคำพูด " หรือ double quotes เสมอ*** เช่น "Abc" และข้อมูลชนิด cha จะอยู่ในเครื่องหมายอัญประกาศเดี่ยวหรือ ' (single quotes) เสมอ เช่น 'A'

### 2.6.1 การประกาศตัวแปร String class

การประกาศตัวแปร String class คล้ายกับการประกาศตัวแปรทั่วไป มักจะประกาศตัวแปรนี้ไว้บนสุดของฟังก์ชั่นหรือตอนต้นของโปรแกรม ตัวแปร String class ที่ถูกประกาศในฟังก์ชั่นหนึ่งจะไม่สามารถเรียกใช้ได้ในอีกฟังก์ชั่นหนึ่ง การกำหนดค่าให้ตัวแปรอาจทำพร้อมกับการประกาศตัวแปรก็ได้

**รูปแบบ** :

String var;

**โดยที่** var คือ ชื่อของตัวแปร String class มีข้อกำหนดเช่นเดียวกันกับการประกาศตัวแปรทั่วๆไป

**ตัวอย่างที่ 2-23**

```
String a = "IPST";
char data[] = {'I', 'P', 'S', 'T'};
String b = String(data);
void setup()
{
    Serial.begin(9600);      // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    Serial.println(a);       // ส่งค่าของตัวอักษร IPST ออกไปยังพอร์ตอนุกรม
    Serial.println(b);       // ส่งค่าของอะเรย์ตัวอักษร IPST ออกไปยังพอร์ตอนุกรม
}
void loop()
{}
```

## 2.6.2 การกำหนดค่าให้ String class

การกำหนดค่าให้ตัวแปร String class ทำได้โดยใช้เครื่องหมาย = เช่นเดียวกับตัวแปรชนิด char จะกำหนดค่าให้ตัวแปรแบบนี้ได้ก็ต่อเมื่อชนิดข้อมูลเป็น String class เหมือนกัน หรือใช้คำสั่ง String() แปลงข้อมูลให้เป็น string

**รูปแบบ** :

String(data)

**โดยที่** data คือ ข้อมูลชนิด byte[], char[], int หรือ String

เนื่องจากอะเรย์ของตัวอักษร (char) รวมกันเป็น string ได้ และข้อมูลชนิด byte ถอดรหัสเป็นตัวอักษรได้ ดังนั้นอะเรย์ของ byte จึงรวมกันเป็น string ได้เช่นกัน และนำข้อมูลชนิด int มาใช้แทนตัวอักษรพิเศษบางตัว

**<u>ตัวอย่างที่ 2-24</u>**

```
char t[] = "A character string";
String a = String("hello");
String b = String(" there");
String c = String(t);
void setup()
{
    Serial.begin(9600);         // ตั้งค่าอัตราบอดเป็น 9600
    a += b;                     // ตัวแปร a ตอนนี้จะเก็บข้อความ 'hello there'
    Serial.println(c);          // ส่งข้อความ A character string ออกไปยังพอร์ตอนุกรม
    Serial.println(a);          // ส่งข้อความที่เก็บไว้ในตัวแปร a ออกไปยังพอร์ตอนุกรม
}
void loop()
{}
```

### 2.6.3 เมธอดต่างๆใน String class

ในการเข้าถึงเมธอดต่างๆ ใน String class ทำได้โดยนำตัวแปร String class มา . (dot) แล้วตามด้วยฟังก์ชั่นหรือคำสั่งที่ต้องการ สำหรับเมธอดที่ตัวแปร String class ใน Wiring ใช้ได้มีดังนี้

#### 2.6.3.1 length()

เป็นคำสั่งคืนค่าเป็นความยาวหรือจำนวนตัวอักษรของ String

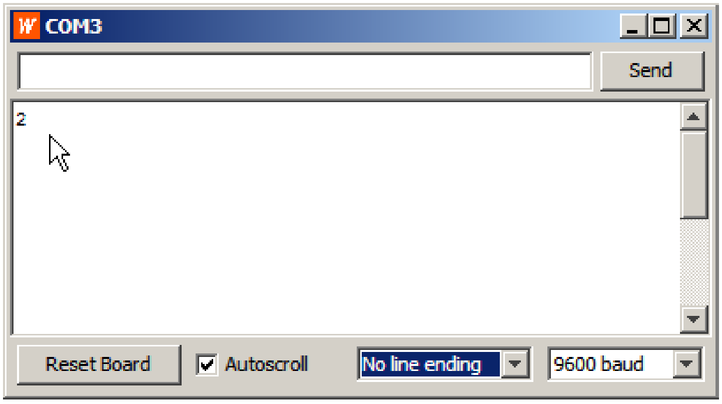
**รูปแบบ :**

string. length();

โดยที่ string คือตัวแปร String class

**ตัวอย่างที่ 2-25**

```
String str1 = "IPST";
void setup()
{
    str1 = "IP";
    int l1 = str1.length();
    Serial.begin(9600);        // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    Serial.println(l1, DEC);   // ที่หน้าต่าง Serial monitor แสดง 2 เท่ากับจำนวนอักษรของตัวแปร str1
}
void loop()
{}
```



### 2.6.3.2 replace()

เป็นคำสั่งแทนที่ตัวอักษรใน string (char1) ด้วยตัวอักษรที่กำหนดไว้ (char1)

**รูปแบบ :**

string.replace(char1, char2)

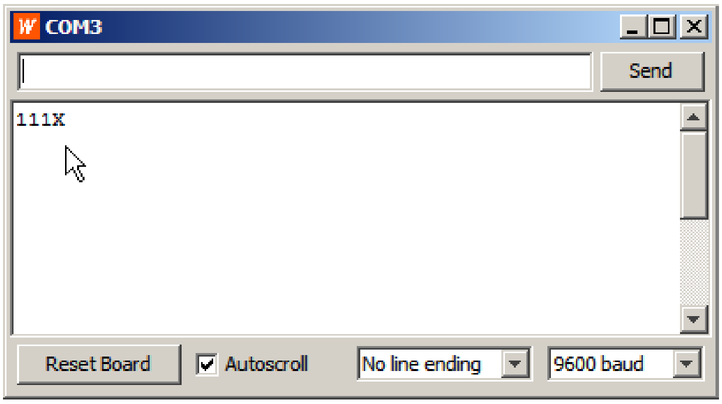
โดยที่ char1 คือ ตัวอักษรที่ต้องการค้นหาเพื่อแทนที่ ต้องเป็นข้อมูลชนิด char

char2 คือ ตัวอักษรที่ต้องการนำไปแทนที่ ต้องเป็นข้อมูลชนิด char

string คือตัวแปร String class

**ตัวอย่างที่ 2-26**

```
String str = "WWWX";
void setup()
{
    Serial.begin(9600);         // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    str.replace('W','1');       // แทนที่อักษร W ทุกตัวด้วยเลข 1
    Serial.println(str);        // ที่หน้าต่าง Serial monitor ของ Wiring IDE จะแสดงข้อความ 111X
}
void loop()
{}
```



### 2.6.3.3 setCharAt()

เป็นคำสั่งแทนที่ตัวอักษรใน string ที่อยู่ในตำแหน่งที่กำหนดไว้ (index) ด้วยตัวอักษรที่กำหนดไว้ (ch) ซึ่งตำแหน่งของ index จะได้รับการคำนวณแบบเดียวกับอะเรย์ กล่าวคือ ถ้า string มีความยาว n ตัวอักษร index จะอยู่ในช่วง 0 ถึง n-1 โดยตัวอักษรแรกสุดคือ 0 ตัวอักษรถัดไปคือ 1

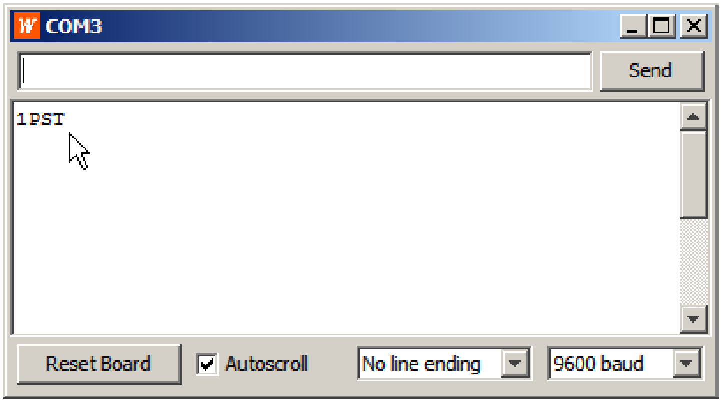
**รูปแบบ :**

string.setCharAt(index, ch);

โดยที่ index คือ ตำแหน่งของตัวอักษรที่ต้องการแทนที่ เป็นตัวเลขชนิด int
ch คือ ตัวอักษรที่ต้องการนำไปแทนที่ ต้องเป็นข้อมูลชนิด char
string คือตัวแปร String class

**<u>ตัวอย่างที่ 2-27</u>**

```
String str = "IPST";
void setup()
{
    Serial.begin(9600);       // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    str.setCharAt(0,'1');     // แทนที่ตัวอักษรตำแหน่ง 0 ด้วย 1
    Serial.println(str);      // ที่หน้าต่าง Serial monitor ของ Wiring IDE จะแสดงข้อความ 1PST
}
void loop()
{}
```



### 2.6.3.4 charAt()

เป็นคำสั่งคืนค่าเป็นตัวอักษรใน string ที่อยู่ในตำแหน่งที่กำหนดไว้ (index)

**รูปแบบ :**

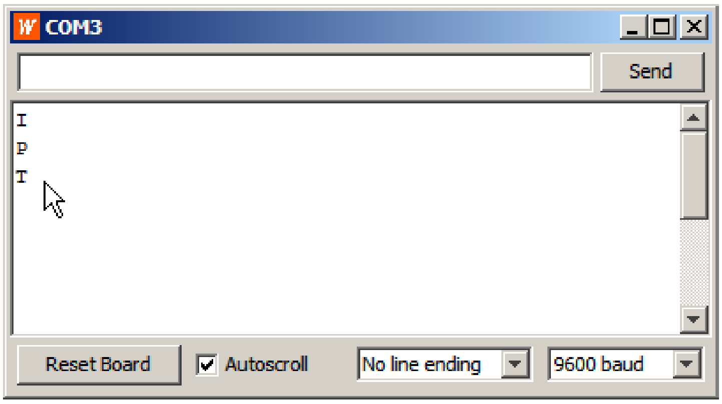
string.charAt(index);

โดยที่ index คือตำแหน่งของตัวอักษร เป็นตัวเลขชนิด int

string คือตัวแปร String class

**<u>ตัวอย่างที่ 2-28</u>**

```
String str = "IPST";
char c1 = str.charAt(0);              // ตัวแปร c1 เก็บตัวอักษรตัวแรก
char c2 = str.charAt(1);              // ตัวแปร c2 เก็บตัวอักษรตัวที่สอง
char c3 = str.charAt(str.length()-1); // ตัวแปร c3 เก็บตัวอักษรสุดท้าย
void setup()
{
    Serial.begin(9600);    // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    Serial.println(c1);    // ส่งข้อมูลของอักษร I ไปยังหน้าต่าง Serial Terminal
    Serial.println(c2);    // ส่งข้อมูลของอักษร P ไปยังหน้าต่าง Serial Terminal
    Serial.println(c3);    // ส่งข้อมูลของอักษร T ไปยังหน้าต่าง Serial Terminal
}
void loop()
{}
```



### 2.6.3.5 indexOf()

เป็นคำสั่งค้นหาตำแหน่ง (index) ของตัวอักษรหรือข้อความที่ค้นหา (str) ที่พบเป็นที่แรกใน string ถ้าหาไม่พบ คำสั่ง indexOf() จะคืนค่าเป็น -1

**รูปแบบ :**

string.indexOf(str);

หรือ

string.indexOf(str, fromIndex);

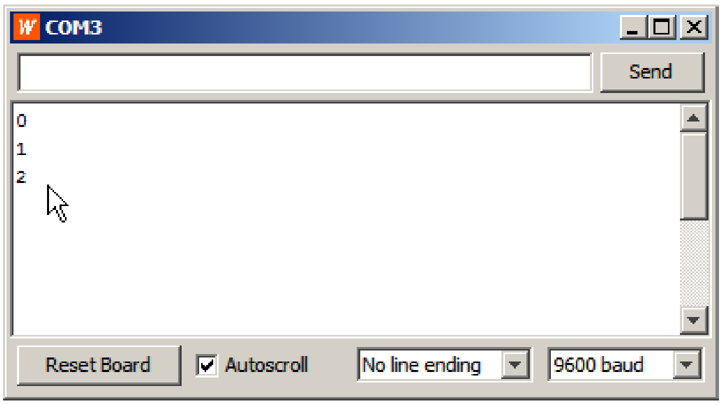
โดยที่ str คือ ตัวอักษรหรือข้อความที่ต้องการค้นหา

fromIndex คือตำแหน่งเริ่มต้นที่ใช้ในการค้นหา เป็นตัวเลขชนิด int

string คือตัวแปร String class

**ตัวอย่างที่ 2-29**

```
String str = "IPST";
int p1 = str.indexOf("I");          // ตัวแปร p1 เก็บตำแหน่งของ I
int p2 = str.indexOf("P");          // ตัวแปร p2 เก็บตำแหน่งของ P
int p3 = str.indexOf("ST");         // ตัวแปร p3 เก็บตำแหน่งของ ST
void setup()
{
    Serial.begin(9600);             // ตั้งค่าอัตราบอดเป็น 9600
    Serial.println(p1, DEC);        // ส่งข้อมูลของเลข 0 ไปยังหน้าต่าง Serial Terminal
    Serial.println(p2, DEC);        // ส่งข้อมูลของเลข 1 ไปยังหน้าต่าง Serial Terminal
    Serial.println(p3, DEC);        // ส่งข้อมูลของเลข 2 ไปยังหน้าต่าง Serial Terminal
}
void loop()
{}
```



### 2.6.3.6 equals()

เป็นคำสั่งเปรียบเทียบ String กับข้อความที่กำหนดไว้ (str) เนื่องจาก string ไม่สามารถเปรียบเทียบด้วยการใช้เครื่องหมาย == ได้ จึงต้องใช้คำสั่งนี้แทน

equals() คืนค่าเป็นจริงเมื่อทั้ง 2 string เหมือนกัน และคืนค่าเป็นเท็จเมื่อทั้ง 2 string ไม่เหมือนกัน ควรระวังเรื่องตัวพิมพ์เล็ก-ใหญ่ในข้อความที่นำมาเปรียบเทียบด้วย

**รูปแบบ** :

string.equals(str);

**โดยที่** str คือ String ที่ต้องการใช้เปรียบเทียบ

string คือตัวแปร String class

**<u>ตัวอย่างที่ 2-30</u>**

```
String str1 = "IPST";
char str3[] = "IPST";
void setup()
{
    Serial.begin(9600);             // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลเป็น 9600
    if(str1.equals(str3) == true)   // ตรวจสอบว่า str1 เหมือนกับ str3 หรือไม่
    {
        Serial.println("Equal");    // ถ้าจริง แสดงข้อความ Equal ที่หน้าต่าง Serial monitor
    }
    else
    {
        Serial.println("Not equal"); // ถ้าไม่จริง แสดงข้อความ Not equal
    }
    if(str1.equals("INEX") == true) // ตรวจสอบว่า str1 เหมือนกับ IPST หรือไม่
    {
        Serial.println("Equal");    // ถ้าจริง ให้แสดงข้อความ Equal ที่หน้าต่าง Serial monitor
    }
    else
    {
        Serial.println("Not equal"); // ถ้าไม่จริง แสดงข้อความ Not equal ที่หน้าต่าง Serial monitor
    }
}
void loop()
{}
```

### 2.6.3.7 startsWith()

**เป็นคำสั่งตรวจสอบว่า String ขึ้นต้นด้วยข้อความที่กำหนดไว้ (str) หรือไม่** คำสั่ง startsWith() จะคืนค่าเป็นจริงหาก string ขึ้นต้นด้วยข้อความที่กำหนดไว้ และคืนค่าเป็นเท็จถ้าไม่ใช่ คำสั่ง startsWith() ต่างกับ equals() ตรงที่ equals() ต้องเปรียบเทียบให้เหมือนทุกตัวอักษรจึงจะคืนค่าเป็นจริง แต่ startsWith() ตรวจสอบเพียงตัวอักษรที่กำหนดไว้

**รูปแบบ :**

string.startsWith(str);

หรือ

string.startsWith(str, offset);

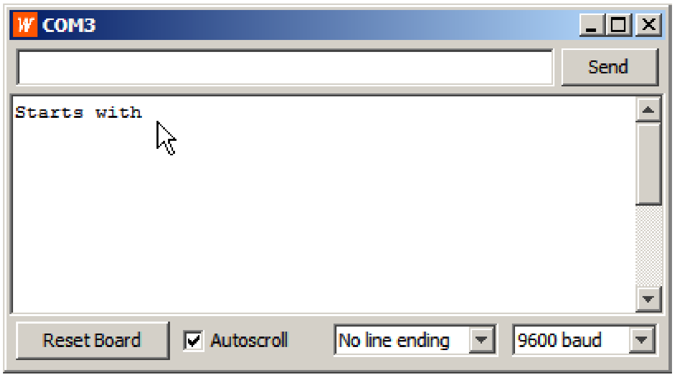
โดยที่ str คือ String ที่ต้องการเปรียบเทียบ

fromIndex คือตำแหน่งเริ่มต้นที่ใช้ในการเปรียบเทียบ เป็นตัวเลขชนิด int

string คือตัวแปร String class

**<u>ตัวอย่างที่ 2-31</u>**

```
void setup()
{
    Serial.begin(9600);                 // ตั้งค่าอัตราบอดเป็น 9600
    String str1 = "IPST";
    String str2 = "IP";
    if(str1.startsWith(str2) == true)   // ตรวจสอบว่า str1 ขึ้นต้นด้วย IN หรือไม่
    {
        Serial.println("Starts with");  // ถ้าจริง แสดงข้อความ Starts with ที่หน้าต่าง Serial monitor
    }
    else
    {
        Serial.println("Not");          // ถ้าไม่เป็นจริง แสดงข้อความ Not ที่หน้าต่าง Serial monitor
    }
}
void loop()
{}
```



### 2.6.3.8 endsWith()

เป็นคำสั่งตรวจสอบว่า String ลงท้ายด้วยข้อความที่กำหนดไว้ (str) หรือไม่ คำสั่ง endsWith() คืนค่าเป็นจริงหาก string ลงท้ายด้วยข้อความที่กำหนดไว้ และคืนค่าเป็นเท็จถ้าไม่ใช่ คำสั่ง endsWith() จะตรวจสอบเพียงตัวอักษรที่กำหนดไว้เท่านั้นเหมือนกับคำสั่ง startsWith()

**รูปแบบ :**

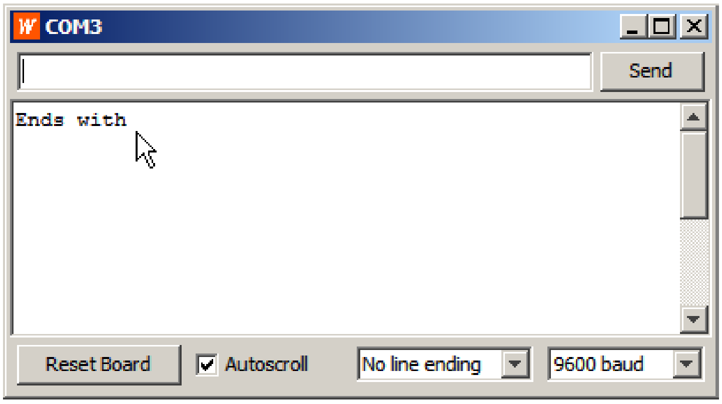
string.endsWith(str);

โดยที่ str คือ String ที่ต้องการจะใช้เปรียบเทียบ

string คือตัวแปร String class

**ตัวอย่างที่ 2-32**

```
void setup()
{
    Serial.begin(9600);                // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลอนุกรมเป็น 9600
    String str1 = "IPST";
    String str2 = "ST";
    if(str1.endsWith(str2) == true)    // ตรวจสอบว่า str1 ลงท้ายด้วย ST หรือไม่
    {
        Serial.println("Ends with");   // ถ้าเงื่อนไขเป็นจริง แสดงข้อความนี้ที่หน้าต่าง Serial monitor
    }
    else
    {
        Serial.println("Not");         // ถ้าไม่เป็นจริง จะแสดงข้อความนี้ที่หน้าต่าง Serial monitor
    }
}
void loop()
{}
```



### 2.6.3.9 substring()

เป็นคำสั่งสร้างตัวอักษรใหม่จากตัวอักษรในตัวแปรเดิม โดยคัดลอกตัวแปร string ต้นฉบับจากตำแหน่งเริ่มต้น (beginIndex) ที่กำหนดไว้จนถึงตำแหน่งสุดท้ายของตัวแปร string หากไม่ได้กำหนดพารามิเตอร์ endIndex ในกรณีที่มีการกำหนดพารามิเตอร์ endIndex คำสั่ง substring() จะคัดลอกจนถึงตำแหน่งสิ้นสุด (endIndex -1) ที่กำหนดไว้ ซึ่งจะน้อยกว่าตำแหน่งที่สิ้นสุดที่กำหนดไว้ 1 ตำแหน่ง

คำสั่ง substring() คืนค่าเป็น string ใหม่

**รูปแบบ :**

string.substring(beginIndex);

หรือ

string.substring(beginIndex, endIndex);

โดยที่ beginIndex คือตำแหน่งเริ่มต้นในการสร้างตัวแปร string ใหม่ เป็นตัวเลขชนิด int

endIndex คือตำแหน่งสิ้นสุดในการสร้างตัวแปร string ใหม่ เป็นตัวเลขชนิด int

string คือตัวแปร String class

**ตัวอย่างที่ 2-33**

```
void setup()
{
    Serial.begin(9600);                  // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลอนุกรมเป็น 9600
    String str1 = "IPST";
    String str2 = "Rabbit";
    String ss1 = str1.substring(2);      // ตัวแปร ss1 เก็บค่า ST
    String ss2 = str2.substring(3);      // ตัวแปร ss2 เก็บค่า bit
    String ss3 = str1.substring(0,1);    // ตัวแปร ss3 เก็บค่า IP
    String result = ss1;
    result += ':';
    result += ss2;
    result += ':';
    result += ss3;
    Serial.println(result);              // แสดงข้อความ ST:bit:IP ที่หน้าต่าง Serial monitor
}
void loop()
{}
```

### 2.6.3.10 toCharArray()

เป็นคำสั่งคืนค่าเป็นอะเรย์ของตัวแปรชนิด char ซึ่งประกอบด้วย ตัวอักษรของ String ซึ่งก็คือการแปลง String class เป็นอะเรย์ของตัวแปรชนิด char คำสั่ง toCharArray() แตกต่างกับคำสั่ง getBytes() ตรงที่คำสั่ง getBytes() จะได้ผลลัพธ์เป็นข้อมูลชนิด byte แต่คำสั่ง toCharArray() จะได้ผลลัพธ์เป็นข้อมูลชนิด char

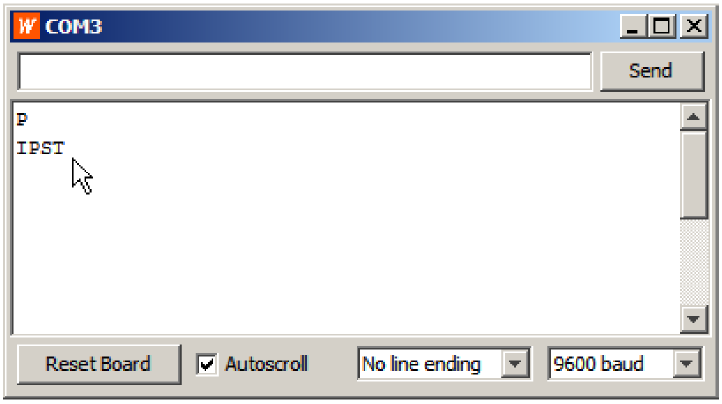
**รูปแบบ :**

string.toCharArray();

โดยที่ string คือตัวแปร String class

**ตัวอย่างที่ 2-34**

```
void setup()
{
    Serial.begin(9600);              // ตั้งค่าอัตราบอดเป็น 9600
    String str = "IPST";
    char array [5];
    str.toCharArray(array,5);
    Serial.println(array[1],BYTE);   // แสดงตัวอักษร P
    Serial.println(array);           // แสดงข้อความ IPST ที่หน้าต่าง Serial monitor
}
void loop()
{}
```



### 2.6.3.11 toLowerCase()

เป็นคำสั่งแปลงตัวอักษรในตัวแปรชนิด string ให้เป็นตัวพิมพ์เล็กทั้งหมด (lower case)

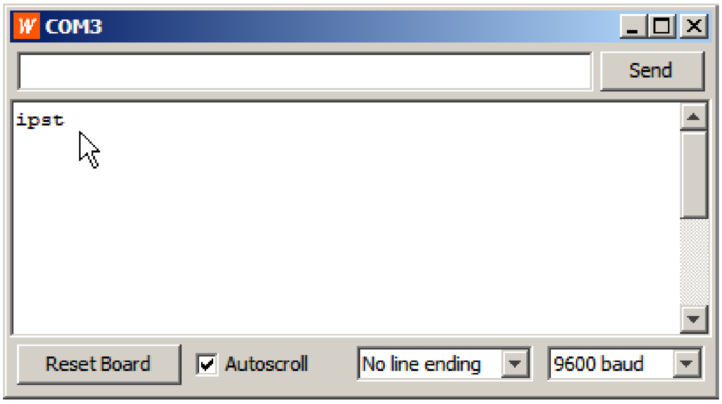
**รูปแบบ :**

string.toLowerCase();

โดยที่ string คือตัวแปร String class

**ตัวอย่างที่ 2-35**

```
void setup()
{
    Serial.begin(9600);          // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    String str1 = "IPST";
    str1.toLowerCase();          // แปลงตัวอักษรเป็นตัวพิมพ์เล็ก
    Serial.println(str1);        // แสดงข้อความ ipst ที่หน้าต่าง Serial Terminal ของ Wiring IDE
}
void loop()
{}
```



### 2.6.3.12 toUpperCase()

เป็นคำสั่งแปลงตัวอักษรในตัวแปรชนิด string ให้เป็นตัวพิมพ์ใหญ่ทั้งหมด (upper case)

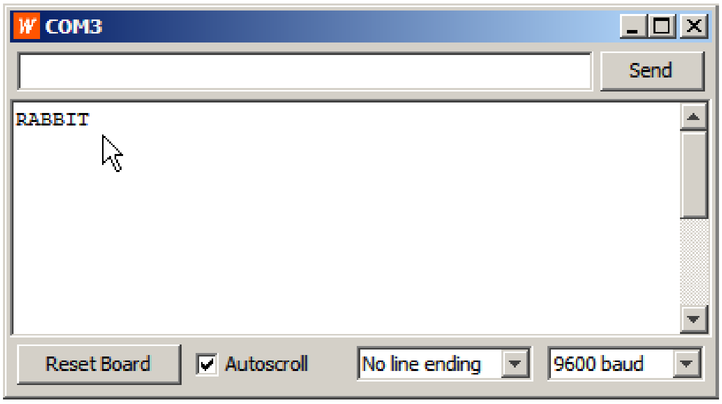
**รูปแบบ :**

string.toUpperCase() ;

โดยที่ string คือตัวแปร String class

**ตัวอย่างที่ 2-36**

```
void setup()
{
    Serial.begin(9600);      // ตั้งค่าอัตราบอดให้กับโมดูลสื่อสารข้อมูลผ่านพอร์ตอนุกรมเป็น 9600
    String str1= “Rabbit”;
    str1.toUpperCase();      // แปลงตัวอักษรเป็นตัวพิมพ์ใหญ่
    Serial.println(str1);    // แสดงข้อความ RABBIT ที่หน้าต่าง Serial Terminal ของ Wiring IDE
}
void loop()
{}
```



## 2.7 ไฟล์ไลบรารีของ Wiring1.0

Wiring1.0 มีไลบรารีเสริมภายนอกจำนวนมาก เพื่อช่วยให้สามารถเขียนโปรแกรมเพื่อติดต่อและใช้งานอุปกรณ์ฟังก์ชั่นพิเศษ ตลอดจนตัวตรวจจับแบบต่างๆ ได้สะดวกและง่ายขึ้น สำหรับชุดกล่องสมองกล IPST-MicroBOX (SE) จะใช้ไฟล์ไลบรารีเสริมที่ชื่อว่า **ipst.h** อันเป็นไฟล์ไลบรารีที่ผู้จัดทำชุดกล่องสมอง IPST-MicroBOX (SE) พัฒนาขึ้นเป็นพิเศษ เพื่ออำนวยความสะดวกในการพัฒนาโปรแกรมควบคุมแผงวงจร IPST-SE ให้ง่ายและมีประสิทธิภาพ ซึ่งมีรายละเอียดอยู่ในคู่มือเริ่มต้นใช้งานชุดกล่องสมองกล IPST-MicroBOX (SE) ซึ่งจัดมาพร้อมในชุดแล้ว

สำหรับไลบรารีเสริมอื่นๆ ที่ Wiring1.0 เตรียมมาให้ใช้งาน สรุปได้ดังนี้

1. **EEPROM.h** ไฟล์ไลบรารีสำหรับการจัดการหน่วยความจำข้อมูลอีอีพรอมภายในไมโครคอนโทรลเลอร์ ATmega644P ที่ใช้ในแผงวงจร IPST-SE

2. **Servo.h** ไฟล์ไลบรารีสำหรับติดต่อและขับเซอร์โวมอเตอร์

3. **LiquidCrystal.h** ไฟล์ไลบรารีติดต่อกับโมดูล LCD

4. **Encoder.h** ไฟล์ไลบรารีสำหรับการใช้งานวงจรเข้ารหัสขั้นพื้นฐาน (ต้องใช้งานร่วมกับอุปกรณ์เข้ารหัส)

5. **Wire.h** ไฟล์ไลบรารีสำหรับติดต่อกับอุปกรณ์ระบบบัส 2 สายและบัส $I^2C$

6. **Matrix.h** ไฟล์ไลบรารีสำหรับใช้งาน LED ด็อตเมตริกซ์ (ต้องการอุปกรณ์เฉพาะ)

7. **Sprite.h** ไฟล์ไลบรารีสำหรับทำภาพเคลื่อนไหวบน LED ดอตเมตริกซ์ (ต้องการอุปกรณ์เฉพาะ)

8. **SoftwareSerial.h** ไฟล์ไลบรารีสื่อสารข้อมูลอนุกรมระหว่าง Wiring กับอุปกรณ์ภายนอกผ่านทางขาพอร์ตดิจิตอลปกติ

9. **Stepper.h** ไฟล์ไลบรารีติดต่อกับวงจรขับสเต็ปเปอร์มอเตอร์ (ต้องการ อุปกรณ์เฉพาะ)

10. **OneWire.h** ไฟล์ไลบรารีสำหรับติดต่อกับอุปกรณ์บัสหนึ่งสาย (เป็นไลบรารีเสริม)

# บทผนวก

# แนะนำชุดกล่องสมองกล IPST MicroBOX SE สำหรับเรียนรู้-ทดลองและพัฒนาโครงงานวิทยาศาสตร์ด้วยไมโครคอนโทรลเลอร์

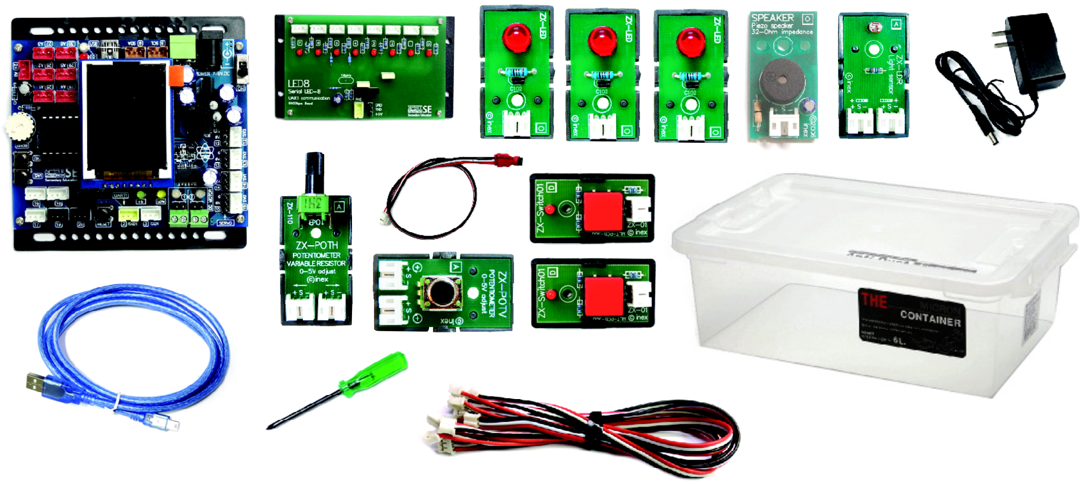
**IPST-MicroBOX (SE)** เป็นชุดแผงวงจรอเนกประสงค์ที่ใช้อุปกรณ์ควบคุมแบบโปรแกรมได้ขนาดเล็กที่เรียกว่า "ไมโครคอนโทรลเลอร์" (microcontroller) ทำงานร่วมกับวงจรเชื่อมต่อคอมพิวเตอร์เพื่อการโปรแกรมและสื่อสารข้อมูล โดยในชุดประกอบด้วย แผงวงจรควบคุมหลัก IPST-SE ซึ่งมีไมโครคอนโทรลเลอร์เป็นอุปกรณ์หลัก, กลุ่มของแผงวงจรอุปกรณ์แสดงผลการทำงานหรืออุปกรณ์เอาต์พุต อาทิ แผงวงจรแสดงผลด้วยไดโอดเปล่งแสง 8 ดวง และแผงวงจรแสดงผลด้วยไดโอดเปล่งแสงแบบตัวเดี่ยว รวมถึงแผงวงจรอุปกรณ์ตรวจจับสัญญาณหรือเซนเซอร์ (sensor) ซึ่งมีด้วยกันหลากหลายรูปแบบ จึงทำให้ผู้ใช้งานสามารถนำชุดกล่องสมองกล IPST-MicroBOX(SE) นี้มาใช้ในการเรียนรู้, ทดลองและพัฒนาโครงงานทางวิทยาศาสตร์ที่เกี่ยวข้องกับระบบควบคุมอัตโนมัติได้อย่างสะดวกและมีประสิทธิภาพสูง

**IPST-MicroBOX (SE)** มีด้วยกัน 2 รุ่นคือ

1. **รุ่นมาตรฐาน 1** ในชุดนี้ประกอบด้วย แผงวงจรควบคุมหลัก IPST-SE เป็นอุปกรณ์หลักที่มีโมดูลแสดงผลกราฟิก LCD สีในตัว, แผงวงจร LED, แผงวงจรลำโพง, แผงวงจรตรวจจับสัญญาณหรือเซนเซอร์ (sensor) พื้นฐาน, และเครื่องจ่ายไฟ ทำให้นำชุด IPST-MicroBOX (SE) นี้ไปใช้ในการเรียนรู้และเขียนโปรแกรมเพื่อพัฒนาเป็นโครงงานทางวิทยาศาสตร์ที่มีการควบคุมด้วยระบบอัตโนมัติโดยใช้โปรแกรมภาษา C/C++ ในเบื้องต้นได้ภายใต้งบประมาณที่เหมาะสม

2. **รุ่นมาตรฐาน 2** ในชุดประกอบด้วยอุปกรณ์หลักเหมือนกับชุด IPST-MicroBOX (SE) รุ่นมาตรฐาน 1 มีการเพิ่มตัวตรวจจับแสงสะท้อนอินฟราเรดอีก 2 ตัว, มอเตอร์ไฟตรงพร้อมชุดเฟืองขับหรือ DC motor gearbox และชิ้นส่วนทางกลที่จำเป็น เพื่อให้ผู้ใช้งานสามารถต่อยอดการเรียนการสอน การใช้งาน IPST-MicroBOX (SE) นี้ไปสร้างเป็นหุ่นยนต์อัตโนมัติแบบโปรแกรมได้ ทั้งยังรองรับกิจกรรมการแข่งขันได้เป็นอย่างดี

## รายการอุปกรณ์ของชุดกล่องสมองกล IPST-MicroBOX (SE) รุ่นมาตรฐาน 1



ประกอบด้วย

1. แผงวงจรควบคุมหลัก IPST-SE
2. แผงวงจร LED เดี่ยวพร้อมสายสัญญาณ 3 ชุด
3. แผงวงจร LED 8 ดวงพร้อมสายสัญญาณ 1 ชุด
4. แผงวงจรลำโพงเปียโซพร้อมสายสัญญาณ 1 ชุด
5. แผงวงจรสวิตช์พร้อมสายสัญญาณ 2 ชุด
6. แผงวงจรตรวจจับแสงพร้อมสายสัญญาณ 1 ชุด
7. แผงวงจรตัวต้านทานปรับค่าได้พร้อมสายสัญญาณ 2 ชุด
8. ไอซีวัดอุณหภูมิพร้อมสายต่อ 1 ชุด
9. อะแดปเตอร์ไฟตรง +9V 1A
10. สายเชื่อมต่อ USB-miniB สำหรับดาวน์โหลดโปรแกรมและสื่อสารข้อมูล
11. ซีดีรอมบรรจุซอฟต์แวร์และตัวอย่างโปรแกรมการทดลอง
12. คู่มือการทดลอง
13. กล่องบรรจุ
14. ไขควง

# รายการอุปกรณ์ของชุดกล่องสมองกล IPST-MicroBOX (SE) รุ่นมาตรฐาน 2

ประกอบด้วย

1. แผงวงจรควบคุมหลัก IPST-SE
2. แผงวงจร LED เดี่ยวพร้อมสายสัญญาณ 3 ชุด
3. แผงวงจร LED 8 ดวงพร้อมสายสัญญาณ 1 ชุด
4. แผงวงจรลำโพงเปียโซพร้อมสายสัญญาณ 1 ชุด
5. แผงวงจรสวิตช์พร้อมสายสัญญาณ 2 ชุด
6. แผงวงจรตรวจจับแสงพร้อมสายสัญญาณ 1 ชุด
7. แผงวงจรตัวต้านทานปรับค่าได้พร้อมสายสัญญาณ 2 ชุด
8. ไอซีวัดอุณหภูมิพร้อมสายต่อ 1 ชุด

9. อะแดปเตอร์ไฟตรง +9V 1A

10. สายเชื่อมต่อ USB-miniB สำหรับดาวน์โหลดโปรแกรมและสื่อสารข้อมูล

11. ซีดีรอมบรรจุซอฟต์แวร์และตัวอย่างโปรแกรมการทดลอง

12. คู่มือการทดลอง

13. กล่องบรรจุ

14. ไขควง

15. แผงวงจรตรวจจับแสงสะท้อนอินฟราเรดพร้อมสายสัญญาณ 2 ชุด

16. มอเตอร์ไฟตรงพร้อมชุดเฟืองขับรุ่น BO2 อัตราทด 48:1 พร้อมสายเชื่อมต่อแบบ IDC จำนวน 2 ตัว

17. ล้อพลาสติกกลมสำหรับชุดเฟืองขับมอเตอร์และยาง จำนวน 2 ชุด

18. ไขควงแบน 2 มม. สำหรับขันขั้วต่อแบตเตอรี่

19. ตัวขันสกรูพลาสติก

20. แผ่นกริดขนาด 80 x 60 เซนติเมตรและ 80 x 80 เซนติเมตร จำนวน 2 ชุด

21. แผ่นฐานกลมพร้อมล้ออิสระ 1 แผ่น

22. แผ่นฐานกลมสำหรับทำโครงหุ่นยนต์ 1 แผ่น

23. ชิ้นต่อ/แท่งต่อพลาสติกและเสารองพลาสติก

24. ชุดเสารองโลหะ, นอตและสกรู

25. กะบะถ่าน AA 6 ก้อน พร้อมสายและหัวต่อป้องกันการกลับขั้วสำหรับต่อกับแผงวงจรหลัก

26. แผ่นทดสอบการเคลื่อนที่ตามเส้นของหุ่นยนต์

IPST MicroBOX SE
Secondary Education