บทที่ 2

การเขียนโปรแกรมภาษาซีสำหรับไมโครคอนโทรลเลอร์

หัวข้อเรื่อง

2.1 ขั้นตอนในการพัฒนาโปรแกรม

2.2 ภาษาซีสำหรับไมโครคอนโทรลเลอร์

2.3 การสร้างฟังก์ชั่นรองขึ้นใช้เอง

สาระสำคัญ

การเขียนโปรแกรมควบคุมการทำงานไมโครคอนโทรลเลอร์ด้วยภาษาซี เป็นการเขียนโปรแกรมที่มีความยืดหยุ่นมากกว่าและสามารถพัฒนางานได้เร็วกว่าการเขียนโปรแกรมด้วยภาษาแอสเซมบลี การเขียนโปรแกรมเพื่อควบคุมการทำงานของ Arduino ยึดหลักวิธีการเขียนตามไวยากรณ์ภาษาซี ดังนั้นเมื่อสามารถเขียนโปรแกรมควบคุมการทำงานไมโครคอนโทรลเลอร์ Arduino นี้ได้ก็สามารถนำความรู้ไปเขียนโปรแกรมภาษาซีไมโครคอนโทรลเลอร์อื่น ๆ ได้

สมรรถนะประจำหน่วยการเรียนรู้

แสดงความรู้เกี่ยวกับการเขียนผังงาน (Flow chart) การตรวจสอบความถูกต้องของการเขียนโปรแกรมจากผังงาน และสามารถเขียนโปรแกรมจากผังงาน

จุดประสงค์การเรียนรู้

จุดประสงค์ทั่วไป

1. เพื่อให้มีความรู้เกี่ยวกับขั้นตอนในการพัฒนาโปรแกรม
2. เพื่อให้มีความรู้เกี่ยวกับภาษาซีสำหรับไมโครคอนโทรลเลอร์
3. เพื่อให้มีความรู้เกี่ยวกับการสร้างฟังก์ชั่นรองขึ้นใช้เอง

จุดประสงค์เชิงพฤติกรรม

1. บอกเกี่ยวกับขั้นตอนในการพัฒนาโปรแกรมได้
2. บอกเกี่ยวกับภาษาซีสำหรับไมโครคอนโทรลเลอร์ได้
3. บอกวิธีการสร้างฟังก์ชั่นรองขึ้นใช้เองได้
4. ทำแบบฝึกหัดเสร็จทันเวลาและทำแบบทดสอบผ่านเกณฑ์ที่กำหนด

การเขียนโปรแกรมภาษาซีสำหรับไมโครคอนโทรลเลอร์

การใช้งานไมโครคอนโทรลเลอร์นอกจากจะต้องมีวงจรไมโครคอนโทรลเลอร์และมีส่วนของวงจรเชื่อมต่อกับอุปกรณ์ภายนอกเพื่อใช้ในการควบคุมงานต่าง ๆ ตามต้องการที่เรียกว่าฮาร์ดแวร์ (Hardware) แล้วจำเป็นต้องมีชุดคำสั่งหรือโปรแกรมไว้สำหรับสั่งงานให้ไมโครคอนโทรลเลอร์ทำงานตามที่ต้องการซึ่งเรียกว่าซอฟท์แวร์ (Software) ในบทนี้เป็นการเรียนรู้หลักการเขียนโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์

2.1 ขั้นตอนในการพัฒนาโปรแกรม

โปรแกรมคอมพิวเตอร์ที่นำมาใช้งานได้นั้น ไม่สามารถเริ่มต้นจากการเขียนคำสั่งด้วยภาษาคอมพิวเตอร์ได้ทันที จะต้องมีการวิเคราะห์ วางแผน และปฏิบัติตามกระบวนการทำงาน ซึ่งแบ่งออกเป็น 5 ขั้นตอน คือ

1. **วิเคราะห์ปัญหา** โดยจะเริ่มจากการวิเคราะห์ผลลัพธ์ที่ต้องการ (Output) แล้วย้อนกลับไปยังข้อมูลที่นำเข้าสู่ระบบ (Input) ตลอดจนข้อมูลอื่น ๆ ที่เกี่ยวข้องในการที่จะนำไปใช้ในการประมวลผล
2. **ออกแบบวิธีการแก้ปัญหา** เมื่อทราบผลลัพธ์ที่ต้องการและข้อมูลที่นำเข้าสู่ระบบแล้ว ต้องกำหนดการวางแผนในการแก้ปัญหา โดยใช้วิธีเขียนลำดับขั้นตอนการแก้ปัญหาที่เรียกว่าอัลกอริทึม (Algorithm) และใช้เครื่องมือสำหรับช่วยในการเขียนอัลกอริทึมเช่น การเขียนรหัสจำลอง (Pseudo code) การเขียนผังงาน (Flowchart) เป็นต้น
3. **เขียนโปรแกรม** เลือกภาษาคอมพิวเตอร์ที่เหมาะสม โดยพิจารณาจากความสามารถของผู้เขียนโปรแกรมและประสิทธิภาพของภาษาคอมพิวเตอร์นั้น ๆ ให้เหมาะสมกับระบบงานที่ต้องการแล้วเขียนชุดคำสั่งเป็นภาษาคอมพิวเตอร์ตามอัลกอริทึมที่ได้ออกแบบไว้
4. **ทดสอบและแก้ไขโปแกรม** ภายหลังจากเขียนโปรแกรมเสร็จสิ้น จะต้องทำการทดสอบโปรแกรมเพื่อหาข้อผิดพลาด (Error) ซึ่งข้อผิดพลาดที่พบในขั้นตอนการทดสอบโปรแกรมนั้นจะต้องนำมาปรับปรุงแก้ไขโปรแกรมเพื่อให้สามารถทำงานได้ตามต้องการ
5. **จัดทำเอกสารประกอบ** เมื่อโปรแกรมผ่านการทดสอบแล้วก็จะต้องจัดทำเอกสารประกอบซึ่งมีรายละเอียดของวิธีการใช้งานโปรแกรม วิธีการติดตั้งโปรแกรม ตลอดจนขั้นตอนในการพัฒนาโปรแกรม รวมถึงอัลกอริทึมและโปรแกรมต้นฉบับ (Source code) เพื่อประโยชน์ในกรณีที่ต้องการแก้ไขหรือปรับปรุงโปรแกรมภายหลัง[1]

[1]ดร. ศรีไพร ศักดิ์รุ่งไพรศาลกุล “หลักการเขียนโปรแกรม” http://www.gotoknow.org/blogs/posts/269964

ขั้นตอนวิธี หรือ Algorithm (ภาษาไทย : อัลกอริทึม) หมายถึงกระบวนการแก้ปัญหาที่สามารถเข้าใจได้ มีลำดับหรือวิธีการในการแก้ไขปัญหาใดปัญหาหนึ่งอย่างเป็นขั้นเป็นตอนและชัดเจน เมื่อนำเข้าอะไรแล้วจะต้องได้ผลลัพธ์เช่นไร ซึ่งแตกต่างจากการแก้ปัญหาแบบสามัญสำนึกหรือฮิวริสติก (Heuristic) โดยทั่วไปขั้นตอนวิธี จะประกอบด้วย วิธีการเป็นขั้น ๆ และมีส่วนที่ต้องทำแบบวนซ้ำ หรือเวียนเกิดโดยใช้ตรรกะ และ/หรือ ในการเปรียบเทียบในขั้นตอนต่าง ๆ จนกระทั่งเสร็จสิ้นการทำงาน

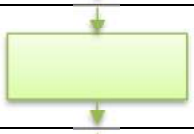
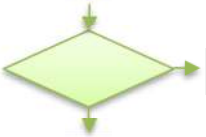
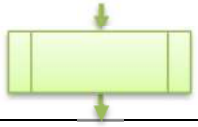
ในการทำงานอย่างเดียวกันเราอาจจะเลือกขั้นตอนวิธีที่ต่างกันเพื่อแก้ปัญหาได้โดยที่ผลลัพธ์ที่ได้ในขั้นสุดท้ายจะออกมาเหมือนกันหรือไม่ก็ได้ และจะมีความแตกต่างที่จำนวนและชุดคำสั่งที่ใช้ต่างกัน ซึ่งส่งผลให้เวลาและขนาดหน่วยความจำที่ต้องการต่างกัน หรือเรียกได้อีกอย่างว่ามีความซับซ้อนต่างกัน

การนำขั้นตอนวิธีไปใช้ไม่จำกัดเฉพาะการเขียนโปรแกรมคอมพิวเตอร์ แต่สามารถนำไปใช้กับปัญหาอื่น ๆ ได้เช่น การออกแบบวงจรไฟฟ้า, การทำงานเครื่องจักรกล, หรือแม้กระทั่งปัญหาในธรรมชาติ เช่น วิธีของสมองมนุษย์ในการคิดเลข หรือวิธีการขนอาหารของแมลง[2]

ผังงาน (Flowchart) คือ รูปภาพ (Image) หรือสัญลักษณ์(Symbol) ที่ใช้เขียนแทนขั้นตอน คำอธิบาย ข้อความ หรือคำพูดที่ใช้ในอัลกอริทึม (Algorithm) เพราะการนำเสนอขั้นตอนของงานให้เข้าใจตรงกัน ระหว่างผู้เกี่ยวข้อง ด้วยคำพูด หรือข้อความทำได้ยากกว่า[3]

ผังงานในการเขียนโปรแกรมเป็นรูปทรงเลขาคณิต ที่บรรจุรายละเอียดกระบวนการประมวลผล โดยมีรูปทรงในการใช้งานหลัก ๆ (เฉพาะงานไมโครคอนโทรลเลอร์) ดังนี้

ตารางที่ 2-1 ผังงานหลักที่ใช้งานไมโครคอนโทรลเลอร์

รูปทรง	ชนิดการประมวลผล	ตัวอย่างเส้นทาง	
	จุดเริ่มต้นหรือจุดสิ้นสุดของโปรแกรม		
	กระบวนการประมวลผล		
	การตัดสินใจ		
	ชุดกระบวนการที่เตรียมไว้แล้ว (โปรแกรมย่อย)		

[2] http://www.com5dow.com

[3] http://158.108.203.7/student/simple/?t46.html

ตารางที่ 2-1 ผังงานหลักที่ใช้งานไมโครคอนโทรลเลอร์ (ต่อ)

รูปทรง	ชนิดการประมวลผล	ตัวอย่างเส้นทาง	
	จุดเชื่อมในหน้าเดียวกัน		
	จุดเชื่อมในหน้าอื่น		

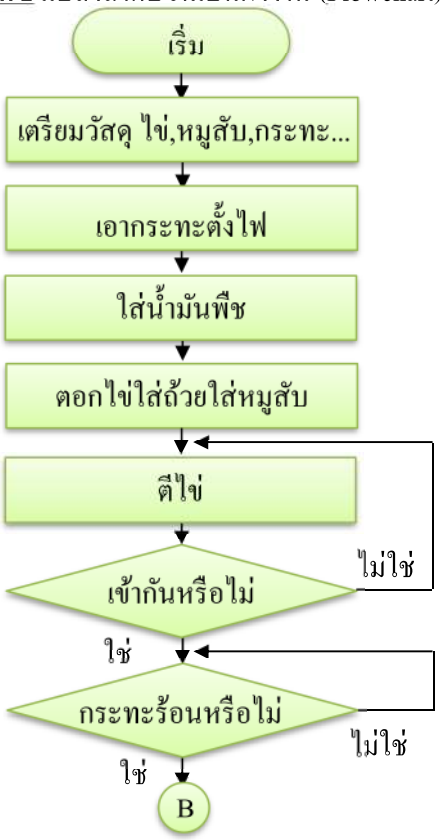
ตัวอย่างการเขียนลำดับขั้นขั้นตอนวิธี และการแปลงเป็นผังงาน

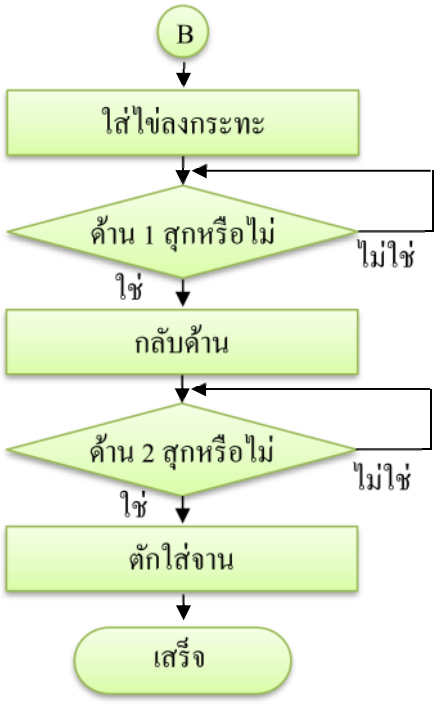
ยกตัวอย่างในชีวิตประจำวันเพื่อให้ง่ายต่อการจินตนาการ โจทย์การทำไข่เจียวหมูสับ

ขั้นที่ 1 เขียนลำดับขั้น (Algorithm)

1. เตรียมวัสดุ ไข่,หมูสับ,กระทะ,เตาไฟ,น้ำมันพืช ,ถ้วย,จาน
2. เอากระทะตั้งไฟ (อุ่นกระทะ)
3. ใส่น้ำมันพืช
4. ตอกไข่ใส่ถ้วยใส่หมูสับ
5. ตีจนเข้ากัน
6. ถ้ากระทะร้อนใส่ไข่ลงไป
7. รอให้หนึ่งด้านสุก
8. กลับด้าน
9. รอให้ด้านที่ 2 สุก
10. ตักออกใส่จาน (เสร็จ)

ขั้นที่ 2 แปลงลำดับขั้นเป็นผังงาน (Flowchart)





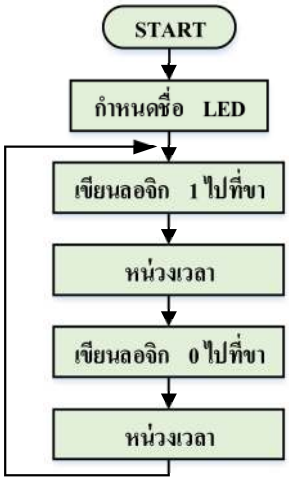
ตัวอย่างการเขียนโปรแกรมไมโครคอนโทรลเลอร์

ตัวอย่างจงเขียนโปรแกรมไฟกระพริบ LED ที่ต่ออยู่ที่ขา D13

ขั้นที่ 1 เขียนลำดับขั้น (Algorithm)

1. กำหนดชื่อ LED กับขาที่เชื่อมต่อ
2. กำหนดโหมดของขาใช้งาน
3. เขียนลอจิก 1 ไปที่ขาเชื่อม LED
4. หน่วงเวลา
5. เขียนลอจิก 2 ไปที่ขาเชื่อม LED
6. หน่วงเวลา
7. กระโดดกลับไปทำในขั้นตอนที่ 3

ขั้นที่ 2 แปลงลำดับขั้นเป็นผังงาน (Flowchart)



ขั้นที่ 3 แปลงผังงานเป็นโปรแกรม

```
#define LED 13
void setup()
{
  pinMode(LED, OUTPUT);
}
void loop()
{
  digitalWrite(LED, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(LED, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
 }
```

2.2 ภาษาซีสำหรับไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ไม่ว่าจะเป็นตระกูลใดก็ตามจะทำงานได้ก็ต่อเมื่อมีชุดคำสั่งที่สั่งให้ทำงานตามที่ต้องการที่เรียกว่าโปรแกรม โดยคำสั่งหรือโปรแกรมที่ไมโครคอนโทรลเลอร์เข้าใจและสามารถทำงานได้อยู่ในรูปของรหัสลอจิก 0 และ 1 หากนำลอจิกมาจับกลุ่มก็เป็นเลขฐาน 16 ที่เรียกว่าภาษาเครื่อง ซึ่งภาษาเครื่องเป็นภาษาที่มนุษย์ไม่สามารถเข้าใจได้เนื่องจากเป็นเลขฐาน 16 ทั้งหมด ดังนั้นในการเขียนโปรแกรมจึงจำเป็นต้องใช้ภาษาที่มนุษย์สามารถเข้าใจได้ โดยภาษาที่มนุษย์เข้าใจได้และใกล้เคียงกับภาษาเครื่องมากที่สุดคือภาษาแอสเซมบลีแต่เนื่องจากการพัฒนางานโดยใช้ภาษาแอสเซมบลีเป็นไปได้ยากและซับซ้อน เพื่อให้ง่ายและรวดเร็วต่อการพัฒนาโปรแกรมใช้งานไมโครคอนโทรลเลอร์ภาษาที่เหมาะสมคือภาษาซี

โครงสร้างของภาษาซี

ภาษาซีเป็นภาษาที่ได้รับความนิยมสูงเป็นภาษาโครงสร้างง่ายต่อการทำความเข้าใจง่ายต่อการนำไปพัฒนาต่อ สามารถเขียนโปรแกรมแยกเป็นส่วน ๆ โดยแต่ละส่วนสามารถเรียกใช้งานได้จากส่วนอื่นของโปรแกรมทำให้สามารถแบ่งงานให้หลายคนไปพัฒนาได้ การเขียนโปรแกรมเป็นส่วน ๆ เรียกว่าฟังก์ชั่น โครงสร้างของภาษาซีมีส่วนประกอบ 2 ส่วนคือ ส่วนหัวโปรแกรมและส่วนตัวโปรแกรม ส่วนตัวโปรแกรมจะมีฟังก์ชั่นหลักชื่อว่า main( ) เพื่อเป็นส่วนหลักในการทำงาน และอาจมีฟังก์ชั่นอื่นที่ผู้ใช้เขียนขึ้นเพื่อใช้งานเรียกว่าฟังก์ชั่นรอง

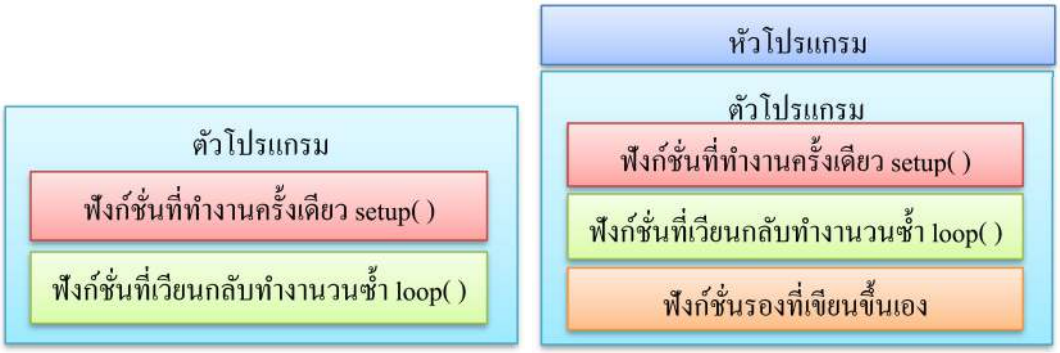
รูปที่ 2-1 โครงสร้างภาษาซี

โครงสร้างของภาษาซีสำหรับ Arduino

โครงสร้างภาษาซีสำหรับ Arduino ถูกจัดใหม่ให้ง่ายต่อผู้ใช้งานเบื้องต้นซึ่งผู้ออกแบบได้จัดวางให้ผู้ใช้งานได้ใช้งานง่ายซึ่งโครงสร้างหลัก ๆ จะมีเพียง 2 ส่วนเท่านั้นคือ

1. setup เป็นส่วนที่เก็บฟังก์ชั่นที่ทำงานครั้งเดียว
2. loop เป็นส่วนที่เก็บฟังก์ชั่นที่เมื่อทำงานครบแล้วจะวนกลับมาทำซ้ำใหม่ตั้งแต่ต้น

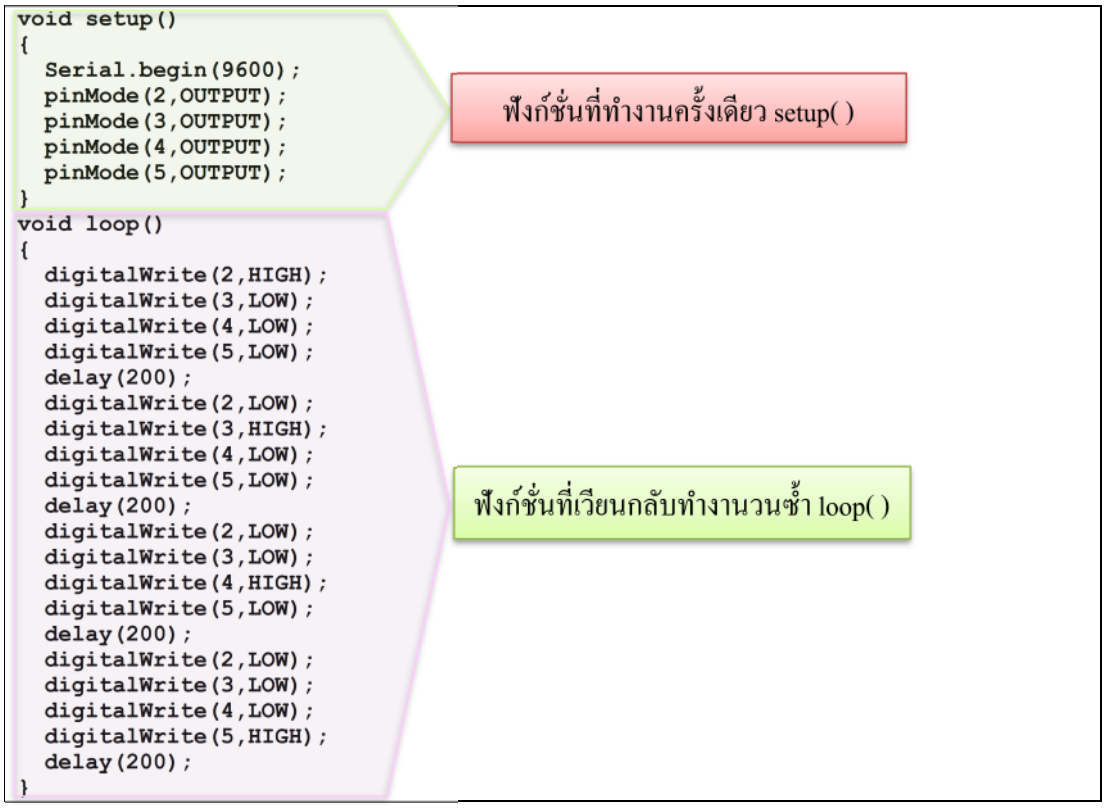
แต่ถ้าต้องการเขียนโปรแกรมขั้นสูงสามารถเขียนในส่วนหัวโปรแกรมและส่วนของฟังก์ชั่นรองที่เขียนขึ้นใช้งานเองเพื่อให้ใช้งานสะดวกมากยิ่งขึ้นได้เช่นเดียวกับภาษาซีมาตรฐาน



(พื้นฐาน) (ขั้นสูง)

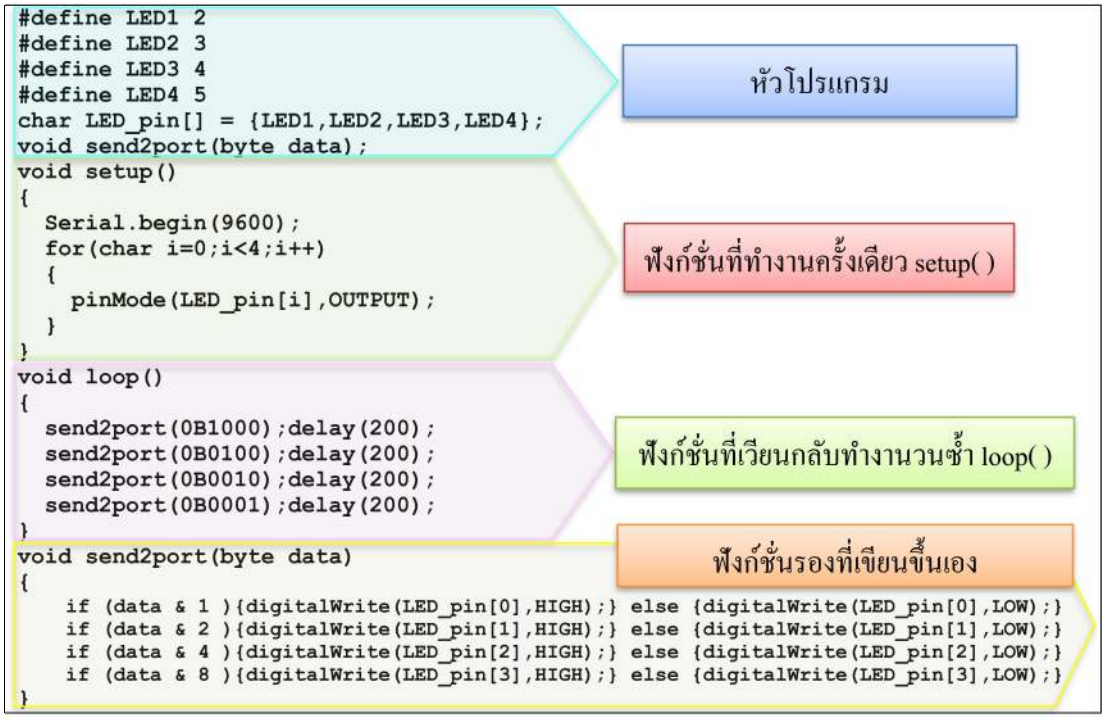
รูปที่ 2-2 โครงสร้างภาษาซีสำหรับ Arduino

ตัวอย่างโปรแกรมที่เขียนด้วยโครงสร้างแบบพื้นฐาน




```
void setup()
{
  Serial.begin(9600);
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
}
void loop()
{
  digitalWrite(2,HIGH);
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  delay(200);
  digitalWrite(2,LOW);
  digitalWrite(3,HIGH);
  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  delay(200);
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);
  digitalWrite(4,HIGH);
  digitalWrite(5,LOW);
  delay(200);
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(5,HIGH);
  delay(200);
}
```

ตัวอย่างโปรแกรมที่เขียนด้วยโครงสร้างแบบขั้นสูง (ที่ให้ผลแบบเดียวกับขั้นพื้นฐาน)




```
#define LED1 2
#define LED2 3
#define LED3 4
#define LED4 5
char LED_pin[] = {LED1,LED2,LED3,LED4};
void send2port(byte data);
void setup()
{
  Serial.begin(9600);
  for(char i=0;i<4;i++)
  {
    pinMode(LED_pin[i],OUTPUT);
  }
}
void loop()
{
  send2port(0B1000);delay(200);
  send2port(0B0100);delay(200);
  send2port(0B0010);delay(200);
  send2port(0B0001);delay(200);
}
void send2port(byte data)
{
    if (data & 1 ){digitalWrite(LED_pin[0],HIGH);} else {digitalWrite(LED_pin[0],LOW);}
    if (data & 2 ){digitalWrite(LED_pin[1],HIGH);} else {digitalWrite(LED_pin[1],LOW);}
    if (data & 4 ){digitalWrite(LED_pin[2],HIGH);} else {digitalWrite(LED_pin[2],LOW);}
    if (data & 8 ){digitalWrite(LED_pin[3],HIGH);} else {digitalWrite(LED_pin[3],LOW);}
}
```

ตัวแปรในภาษาซี

การประกาศตัวแปรคือการจองพื้นที่ในหน่วยความจำเพื่อนำไปใช้งานในฟังก์ชั่น โดยการใช้ชื่อตัวแปรแทนการกำหนดเป็นค่าแอดเดรสของหน่วยความจำ รูปแบบของการประกาศตัวแปรเป็นดังนี้

```
ชนิดของตัวแปร        ชื่อตัวแปร;
```

หรือ

```
ชนิดของตัวแปร        ชื่อตัวแปรที่1,ชื่อตัวแปรที่ 2,... ;
```

หลักการตั้งชื่อตัวแปรในภาษาซี

หลักการตั้งชื่อตัวแปรมีข้อกำหนดหลัก ๆ อยู่ 4 ประการด้วยกันคือ

1. ชื่อที่ตั้งต้องไม่ซ้ำกับคำสงวนของภาษาซี (คำที่ภาษาซีมีใช้งานอยู่แล้ว)
2. การใช้ตัวอักษรใหญ่กับตัวอักษรเล็กถือว่าเป็นคนละตัว
3. ตัวแรกของชื่อตัวแปรต้องเป็นตัวอักษรเท่านั้นตัวถัดไปเป็นตัวเลขได้
4. ชื่อตัวแปรห้ามเว้นวรรค

ชนิดของตัวแปรในภาษาซีสำหรับไมโครคอนโทรลเลอร์ Arduino

ตารางที่ 2-2 ชนิดของตัวแปรในภาษาซีสำหรับไมโครคอนโทรลเลอร์ Arduino

ชนิด	ขนาด		ขอบเขต
	บิต	ไบต์	
boolean	8	1	true, false
char	8	1	-128 ถึง +127
unsigned char	8	1	0 ถึง 255
byte	8	1	0 ถึง 255
int	16	2	-32768 ถึง +32767
unsigned int	16	2	0 ถึง 65535
long	32	4	-2147483648 ถึง +2147483647
unsigned long	32	4	0 ถึง 4294967295
float	32	4	±3.4E±38 (~7 digits)
double	64	8	±1.7E±308 (~15 digits)

พอยน์เตอร์

พอยน์เตอร์เป็นตัวชี้ตำแหน่งข้อมูลของตัวแปรอื่นที่เก็บในหน่วยความจำ โดยตัวพอยน์เตอร์เป็นตัวเก็บตำแหน่งแทนการเก็บข้อมูล ในการใช้งานพอยน์เตอร์จะใช้เครื่องหมาย * นำหน้าและใช้เครื่องหมาย & เมื่อต้องการค่าตำแหน่งของตัวแปรอื่นโดยมีรูปแบบดังนี้

รูปแบบ

```
ชนิดของตัวแปร        * ชื่อตัวแปร;
```

ตัวอย่าง

```
int *n;
int i;
i=10;
n=&i;
```

อาร์เรย์

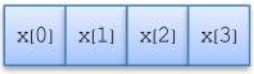
อาร์เรย์เป็นการเพิ่มความสามารถในการเก็บข้อมูลของตัวแปรให้สามารถเก็บเป็นชุดได้ โดยใช้ชื่อตัวแปรเดิมได้ การใช้งานตัวแปรอาร์เรย์จะใช้เครื่องหมาย [ ] ต่อท้ายตัวแปร โดยภายในวงเล็บเป็นตัวเลข ซึ่งสามารถใช้งานได้หลายมิติ มีรูปแบบดังนี้

รูปแบบ

```
ชนิดของตัวแปร  ชื่อตัวแปร[ตัวเลข];                  // เป็นอาร์เรย์ 1 มิติ
ชนิดของตัวแปร  ชื่อตัวแปร[ตัวเลข,ตัวเลข];           // เป็นอาร์เรย์ 2 มิติ
ชนิดของตัวแปร  ชื่อตัวแปร[ตัวเลข,ตัวเลข,ตัวเลข];    // เป็นอาร์เรย์ 3 มิติ
```

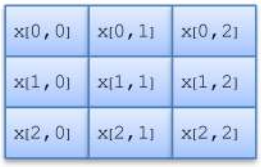
ตัวอย่างอาร์เรย์ 1 มิติ

int x[4];



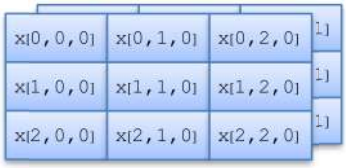
ตัวอย่างอาร์เรย์ 2 มิติ

int x[3,3];



ตัวอย่างอาร์เรย์ 3 มิติ

int x[3,3,2];



ตัวดำเนินการในภาษาซี

ตัวดำเนินการในภาษาซีแบ่งตามลักษณะการกระทำได้ 5 กลุ่มด้วยกันคือ

1. ตัวกระทำทางคณิตศาสตร์
2. ตัวกระทำทางลอจิกระดับบิต
3. ตัวกระทำบูลีน
4. ตัวกระทำเปรียบเทียบ
5. ตัวกระทำประสม

โดยในแต่ละลักษณะมีรายละเอียดดังต่อไปนี้

ตารางที่ 2-3 ตัวกระทำทางคณิตศาสตร์

เครื่องหมาย	การกระทำ	ตัวอย่าง	คำอธิบาย
+	บวก	x=y+z;	x เท่ากับค่าในตัวแปร y บวกกับค่าในตัวแปร z
-	ลบ	x=y-z;	x เท่ากับค่าในตัวแปร y ลบด้วยค่าในตัวแปร z
*	คูณ	x=y*z;	x เท่ากับค่าในตัวแปร y คูณด้วยค่าในตัวแปร z
/	หาร	x=y/z;	x เท่ากับค่าในตัวแปร y หารด้วยค่าในตัวแปร z
%	หารเอาเศษ	x=y%z;	x เท่ากับเศษของการหารระหว่างตัวแปร y กับตัวแปร z

ตารางที่ 2-4 ตัวกระทำทางลอจิก

เครื่องหมาย	การกระทำ	ตัวอย่าง	คำอธิบาย
&	แอนด์	x=y&z;	x เท่ากับค่าในตัวแปร y แอนด์กับค่าในตัวแปร z
|	ออร์	x=y|z;	x เท่ากับค่าในตัวแปร y ออร์กับค่าในตัวแปร z
^	เอ็กซ์คลูซิฟออร์	x=y^z;	x เท่ากับค่าในตัวแปร y เอ็กซ์คลูซิฟออร์กับ z
~	วันคอมพลีเมนต์	x=~y;	x เท่ากับค่าตรงข้ามของค่าในตัวแปร y
<<	เลื่อนไปทางซ้าย	x=x<<1;	เลื่อนข้อมูลใน x ไปทางซ้ายไป 1 บิต
>>	เลื่อนไปทางขวา	x=x>>2;	เลื่อนข้อมูลใน x ไปทางขวาไป 2 บิต

ตารางที่ 2-5 ตัวกระทำบูลีน

เครื่องหมาย	การกระทำ	คำอธิบาย
&&	แอนด์	เชื่อมเงื่อนไข 2 เงื่อนไขด้วยคำว่า “และ”
||	ออร์	เชื่อมเงื่อนไข 2 เงื่อนไขด้วยคำว่า “หรือ”
!	อินเวิร์ส	ใช้ตรวจสอบตัวแปรว่าเท่ากับศูนย์หรือไม่เช่น if (!x)

ตารางที่ 2-6 ตัวกระทำการเปรียบเทียบ

เครื่องหมาย	การกระทำ	ตัวอย่าง	คำอธิบาย
>	มากกว่า	if(x>10)	x มากกว่า 10 ใช่หรือไม่
<	น้อยกว่า	if(x<10)	x น้อยกว่า 10 ใช่หรือไม่
>=	มากกว่าหรือเท่ากับ	if(x>=10)	x มากกว่าหรือเท่ากับ 10 ใช่หรือไม่
<=	น้อยกว่าหรือเท่ากับ	if(x<=10)	x น้อยกว่าหรือเท่ากับ 10 ใช่หรือไม่
= =	เท่ากับ	if(x= =10)	x เท่ากับ 10 ใช่หรือไม่
!=	ไม่เท่ากับ	if(x!=10)	x ไม่เท่ากับ 10 ใช่หรือไม่

ตารางที่ 2-7 ตัวกระทำประสม

เครื่องหมาย	การกระทำ	ตัวอย่าง	คำอธิบาย
++	เพิ่มค่า 1 ค่า	x++;	เพิ่มค่า x ขึ้น 1 ค่า
--	ลดค่า 1 ค่า	x--;	ลดค่า x ลง 1 ค่า
+=	บวก	x+=2;	x ใหม่เท่ากับ x เดิมบวกกับ 2
-=	ลบ	x-=2;	x ใหม่เท่ากับ x เดิมลบด้วย 2
*=	คูณ	x*=2;	x ใหม่เท่ากับ x เดิมคูณด้วย 2
/=	หาร	x/=2;	x ใหม่เท่ากับ x เดิมหารด้วย 2
%=	หารเอาเศษ	x%=2;	x ใหม่เท่ากับ x เดิมหารด้วย 2 แล้วเอาเศษ
&=	แอนด์	x&=2;	x ใหม่เท่ากับ x เดิมแอนด์ด้วย 2
|=	ออร์	x|=2;	x ใหม่เท่ากับ x เดิมออร์ด้วย 2

ไวยากรณ์ภาษาซี

1. **ประกาศชื่อแทน** เป็นการประกาศใช้ชื่ออื่นแทนค่าที่ต้องการเพื่อให้สะดวกต่อการเขียนโปรแกรม รูปแบบเป็นดังนี้

```
#define constantName value
```

constantName: ชื่อที่ต้องการกำหนดตั้ง

value: ค่าที่ต้องการกำหนดให้ชื่อนั้นมีค่าเท่ากับ

ตัวอย่าง

```
#define LED 13
```

หมายถึง กำหนดให้คำว่า LED มีค่าเท่ากับ 13

2. **การรวมไฟล์อื่นเข้ามาร่วม** เป็นการประกาศไฟล์อื่น ๆ เข้ามารวมกับตัวโปรแกรมก่อนการคอมไพล์ รูปแบบเป็นดังนี้

```
#include <file>
```

```
#include "file"
```

file: ชื่อไฟล์ที่ต้องการนำเข้ามารวมกับโค้ดโปรแกรมที่กำลังเขียนขึ้น

ตัวอย่าง

```
#include <SPI.h>
```

หมายถึง ให้โปรแกรมคอมไพลเลอร์ทำการรวมไฟล์ที่ชื่อ SPI.h ก่อนทำการคอมไพล์

3. **การใส่หมายเหตุลงในโค้ดโปรแกรม** เป็นการใส่ข้อความใด ๆ ลงในโค้ดโปรแกรมเพื่อที่จะอธิบายโปรแกรมหรือเพื่อบันทึกความจำว่าโค้ดในตำแหน่งนั้น ๆ เขียนขึ้นเพื่อประสงค์สิ่งใด การใส่หมายเหตุจะต้องใส่เครื่องหมายเพื่อให้คอมไพล์เลอร์ข้ามการคอมไพล์ในส่วนนี้ รูปแบบเป็นดังนี้

```
//.....          ใช้ในกรณีบรรทัดเดียว
```

```
/*.....*/       ใช้ในกรณีหลายบรรทัด
```

4. **การใส่เครื่องหมายท้ายฟังก์ชั่น** ภาษาซีเป็นภาษาที่มีการใส่เครื่องหมายท้ายฟังก์ชั่นซึ่งเป็นสิ่งที่แตกต่างและโดดเด่นกว่าโปรแกรมในภาษาอื่น ๆ โดยมีหลักคิดดังนี้

```
;      สำหรับฟังก์ชั่นที่ทำงานเสร็จสิ้นในตัว
```

```
{ }    สำหรับฟังก์ชั่นที่มีฟังก์ชั่นอื่นรวมเข้าไปด้วย
```

ฟังก์ชั่นการดำเนินการแบบทางเลือก

ในการเขียนโปรแกรมเพื่อให้ไมโครคอนโทรลเลอร์ทำงานในสิ่งที่ต้องการ นอกจากฟังก์ชั่นที่สั่งให้ทำงานเป็นลำดับแล้ว จำเป็นต้องใช้ฟังก์ชั่นที่มีการทำงานแบบให้เลือกเส้นทางการทำงาน โดยการทำตามเงื่อนไข หรือการให้ทำซ้ำแบบมีเงื่อนไขหรือไม่มีเงื่อนไข โดยฟังก์ชั่นที่มีการทำงานแบบทางเลือกในภาษาซีมีด้วยกัน 4 ฟังก์ชั่นคือ

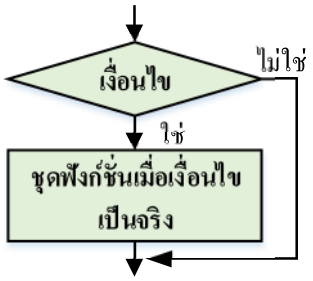
1. ฟังก์ชั่น if
2. ฟังก์ชั่น if-else
3. ฟังก์ชั่น if-else if-else
4. ฟังก์ชั่น switch

ฟังก์ชั่น if (ทางเลือกเดียว)

ฟังก์ชั่น if เป็นฟังก์ชั่นที่มีการตรวจสอบเงื่อนไข โดยถ้าเงื่อนไขเป็นจริงจะทำงานตามชุดฟังก์ชั่นที่กำหนดไว้ รูปแบบเป็นดังนี้

if (เงื่อนไขที่ตรวจสอบ)

ผังงาน	โค้ดโปรแกรม

	`if (conditional)`
	`{`
	`  // put your code here`
	`  // if conditional true`
	`}`
	
	`Example:`
	`if (value<50)`
	`{`
	`  digitalWrite(13,HIGH);`
	`}`

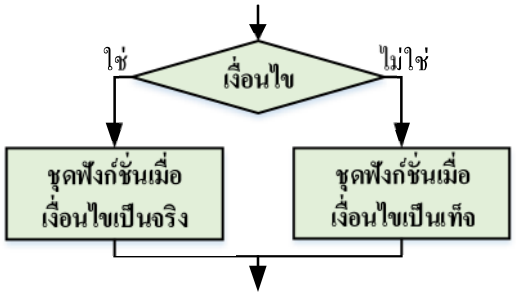
*หากชุดฟังก์ชั่นที่ให้ทำงานเมื่อเงื่อนไขเป็นจริงมีเพียงฟังก์ชั่นเดียว ไม่ต้องใส่วงเล็บปีกกา {...} ก็ได้

ฟังก์ชั่น if-else (สองทางเลือก)

การตรวจสอบเงื่อนไขที่มีชุดฟังก์ชั่นให้ทำงานเมื่อเงื่อนไขเป็นจริงและมีชุดฟังก์ชั่นให้ทำงานเมื่อเงื่อนไขเป็นเท็จ เราจะใช้ฟังก์ชั่น if-else มาใช้งาน มีรูปแบบดังนี้

if (เงื่อนไขที่ตรวจสอบ) else

ผังงาน	โค้ดโปรแกรม

	`if (conditional)`
	`{`
	`  // put main code here`
	`  // if conditional true`
	`}`
	`else`
	`{`
	`  //put main code here`
	`  //if conditional false`
	`}`

ฟังก์ชั่น if-else แบบย่อ

โค้ดโปรแกรมฟังก์ชั่น if-else แบบปกติ	โค้ดโปรแกรมฟังก์ชั่น if-else แบบย่อ

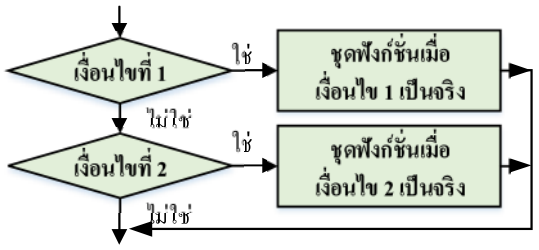
`if (SW1==LOW){`
`  digitalWrite(LED,HIGH);`
`}`
`else{`
`  digitalWrite(LED,LOW);`
`}`
	`digitalWrite(LED,(SW1==LOW)? HIGH:LOW);`

ฟังก์ชั่น if-else if....else (หลายทางเลือก)

เป็นฟังก์ชั่นที่มีการตรวจสอบเงื่อนไขหลายเงื่อนไข และมีชุดฟังก์ชั่นที่เตรียมให้ทำงานในแต่ละเงื่อนไขหากเงื่อนไขนั้น ๆ ถูกต้อง

if (เงื่อนไขที่ตรวจสอบ) else if (เงื่อนไขที่ตรวจสอบ)

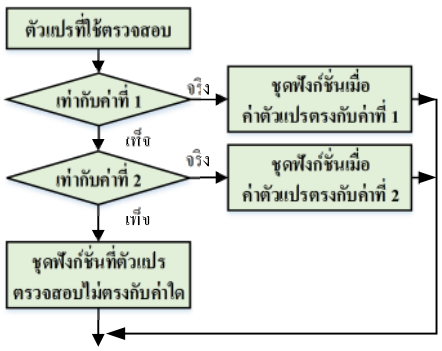
ผังงาน	โค้ดโปรแกรม

	`if (conditional1)`
	`{`
	`  // put main code here`
	`  // if conditional1 true`
	`}`
	`else if (conditional2)`
	
	`{`
	`  //put main code here`
	`  //if conditional2 true`
	`}`

ฟังก์ชั่น switch...case (หลายทางเลือก)

ฟังก์ชั่น switch…case เป็นฟังก์ชั่นหลายทางเลือกอีกฟังก์ชั่นหนึ่งที่มีการทำงานคล้าย ๆ ฟังก์ชั่น if-else if…else ต่างตรงที่การตรวจสอบเงื่อนไขจะใช้การตรวจสอบการเท่ากันของตัวแปรที่ใช้ตรวจสอบเท่านั้น โดยเมื่อตรวจสอบค่าแล้วเท่ากับค่าที่กำหนดให้ทำฟังก์ชั่นที่เตรียมไว้

ผังงาน	โค้ดโปรแกรม

	`switch (variable)`
	`{`
	`  case 1:`
	`  // put code here for case 1`
	`  break;`
	`  case 2:`
	`  // put code here for case 2`
	`  break;`
	`  default:`
	`  // put code here for default`
	`  break;`
	`}`

ฟังก์ชั่นการดำเนินการแบบวนซ้ำ

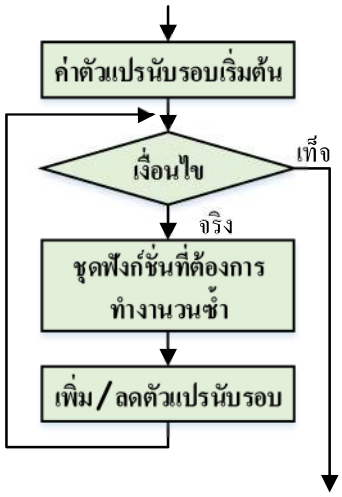
การเขียนโปรแกรมสั่งงานไมโครคอนโทรลเลอร์ ต้องมีการทำงานแบบวนซ้ำหรือวนรอบเพื่อที่จะทำงานในชุดคำสั่งเดิม ลักษณะการทำงานมีทั้งแบบมีเงื่อนไขหรือไม่มีเงื่อนไข ในภาษาซีมีฟังก์ชั่นสั่งงานให้ไมโครคอนโทรลเลอร์ทำงานซ้ำมีดังนี้

1. ฟังก์ชั่น for
2. ฟังก์ชั่น while
3. ฟังก์ชั่น while(1)
4. ฟังก์ชั่น do-while

ฟังก์ชั่น for

ฟังก์ชั่น for เป็นฟังก์ชั่นที่ใช้ในกรณีที่ทราบจำนวนรอบที่จะทำงานซ้ำ โดยมีรูปแบบดังนี้

รูปแบบ	ผังงาน

for(ค่าเริ่มต้น;เงื่อนไข;เพิ่มหรือลดค่า)
{
//ชุดฟังก์ชั่นที่ต้องการทำซ้ำ
}
	

ตัวอย่าง

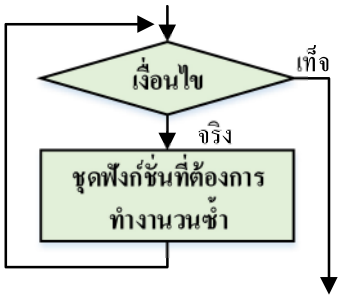
โค้ดโปรแกรม	คำอธิบาย

`for(int i=0;i<10;i++)`
`{`
`digitalWrite(13,HIGH);`
`delay(500);`
`digitalWrite(13,LOW);`
`delay(500);`
`}`
	ประกาศและกำหนดตัวแปรนับรอบเป็นตัวแปร i เป็นตัวแปรชนิด integer โดยให้มีค่าเริ่มต้นเท่ากับศูนย์ ทำวนซ้ำไปเรื่อย ๆ หากค่าตัวแปรยังน้อยกว่า 10 โดยในรอบถัดไปให้เพิ่มค่าในตัวแปรนับรอบขึ้น 1 ค่า

ฟังก์ชั่น while

ฟังก์ชั่น while เป็นฟังก์ชั่นที่ให้ทำงานวนซ้ำหรือวนรอบโดยมีการตรวจสอบเงื่อนไขก่อนถ้าหากเงื่อนไขเป็นจริงจะทำงานตามชุดฟังก์ชั่นที่เตรียมไว้ เมื่อทำงานในชุดฟังก์ชั่นที่เตรียมไว้เสร็จจะมีการวนกลับไปตรวจสอบเงื่อนไขอีก หากเงื่อนไขเป็นจริงจะทำงานในชุดฟังก์ชั่นที่เตรียมไว้โดยทำแบบนี้ไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จจะออกจากวงรอบการทำซ้ำ โดยมีรูปแบบดังนี้

รูปแบบ	ผังงาน

while(เงื่อนไข)
{
//ชุดฟังก์ชั่นที่ต้องการทำซ้ำ
}
	

ตัวอย่าง

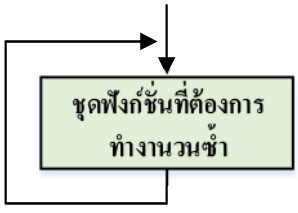
โค้ดโปรแกรม	คำอธิบาย

`i=0;`
`while(i<10)`
`{`
`digitalWrite(13,HIGH);`
`delay(500);`
`digitalWrite(13,LOW);`
`delay(500);`
`i++;`
`}`
	ตรวจสอบก่อนว่าเงื่อนไขเป็นจริงอยู่หรือไม่ (i ยังน้อยกว่า 10) หากเงื่อนไขเป็นจริงให้ทำงานที่เตรียมไว้ เมื่อทำงานครบให้กลับมาตรวจสอบเงื่อนไขใหม่วนซ้ำไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ

ฟังก์ชั่น while(1)

ฟังก์ชั่น while เป็นฟังก์ชั่นที่ให้ทำงานวนซ้ำหรือวนรอบไม่รู้จบ เนื่องจาก 1 คือเป็นจริงตลอดไป ในโปรแกรม Arduino ก็คือฟังก์ชั่น loop( ) นั่นเอง

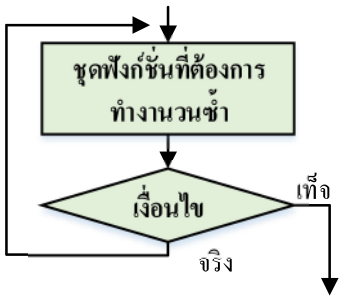
รูปแบบ	ผังงาน

while(1)
{
//ชุดฟังก์ชั่นที่ต้องการทำซ้ำ
}
	

ฟังก์ชั่น do-while

ฟังก์ชั่น do-while เป็นฟังก์ชั่นที่ให้ทำงานวนซ้ำหรือวนรอบ โดยมีการตรวจสอบเงื่อนไขการทำงานคล้ายกับฟังก์ชั่น while ต่างตรงที่ฟังก์ชั่น do-while จะทำงานในชุดฟังก์ชั่นที่เตรียมไว้ทำซ้ำไปก่อน 1 รอบแล้วจึงตรวจสอบเงื่อนไข

รูปแบบ	ผังงาน

do
{
//ชุดฟังก์ชั่นที่ต้องการทำซ้ำ
} while(เงื่อนไข)
	

ตัวอย่าง

โค้ดโปรแกรม	คำอธิบาย

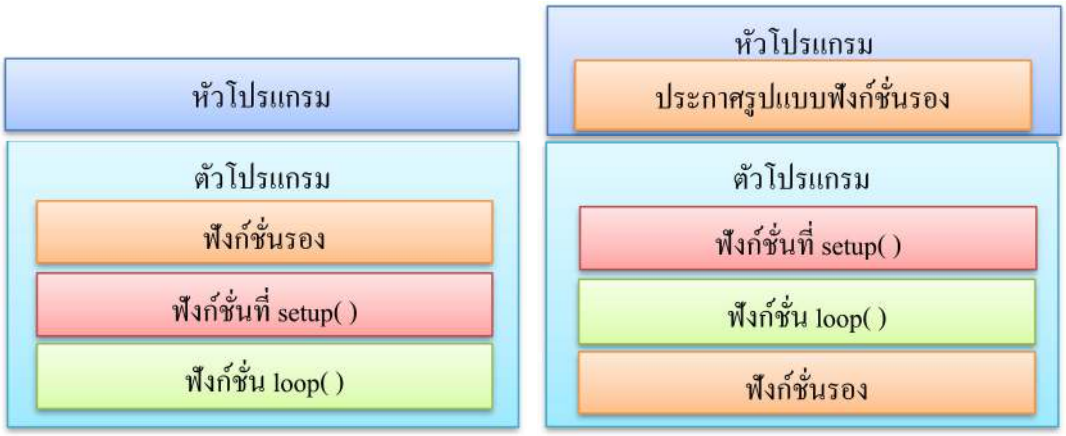
`i=0;`
`do`
`{`
`digitalWrite(13,HIGH);`
`delay(500);`
`digitalWrite(13,LOW);`
`delay(500);`
`i++;`
`} while(i<10)`
	ทำงานในฟังก์ชั่นที่เตรียมไว้ แล้วตรวจสอบเงื่อนไขว่าตัวแปร i ยังมีค่าน้อยกว่า 10 หรือไม่หากยังน้อยกว่าให้วนกลับไปทำใหม่ซ้ำ ๆ จนกว่าเงื่อนไขจะเป็นเท็จ

2.3 การสร้างฟังก์ชั่นรองขึ้นใช้เอง

ฟังก์ชั่นต่าง ๆ ที่ได้กล่าวมาแล้วเป็นฟังก์ชั่นที่ภาษาซีมีให้ใช้งาน แต่ถ้าหากผู้ใช้งานต้องการฟังก์ชั่นที่มีการทำงานตามลักษณะเฉพาะส่วนอย่างใดอย่างหนึ่งที่มีการเรียกใช้งานซ้ำ ๆ หรือเพื่อแยกงานให้เป็นส่วน ๆ ให้ง่ายต่อการเขียนโปรแกรมสามารถเขียนขึ้นเพื่อใช้งานเองได้ การจัดวางตำแหน่งของฟังก์ชั่นรองสามารถทำได้สองแบบคือ

1. **วางก่อนฟังก์ชั่นที่เรียกใช้งาน** ตำแหน่งการวางลักษณะในนี้ไม่ต้องประกาศรูปแบบของฟังก์ชั่น (Prototype) ซึ่งจะต้องวางฟังก์ชั่นรองต่อจากหัวโปรแกรมและก่อนฟังก์ชั่น setup การวางลักษณะนี้มีข้อดีตรงที่ไม่ต้องเขียนประกาศรูปแบบ แต่จะทำให้ฟังก์ชั่นที่ใช้งานหลัก (setup, loop) ถูกดันลงไปท้าย ๆ โปรแกรมซึ่งหากมีฟังก์ชั่นรองหลาย ๆ ฟังก์ชั่นยิ่งถูกดันไปล่าง ๆ ของโปรแกรมมากขึ้น หากโปรแกรมมีฟังก์ชั่นรองหลายสิบหลายร้อยบรรทัดจะส่งผลให้หาโปรแกรมหลักเพื่อเขียนโปรแกรมยากขึ้นการแก้ไขโปรแกรมก็จะยากขึ้นตาม

2. **วางหลังฟังก์ชั่นที่เรียกใช้งาน** ตำแหน่งการวางลักษณะนี้จะต้องประกาศรูปแบบของฟังก์ชั่น (Prototype) ซึ่งจะต้องมีเครื่องหมาย ; ปิดท้ายด้วย โดยประกาศไว้ในท้ายของส่วนหัวโปรแกรม การวางลักษณะนี้มีข้อดีตรงที่ฟังก์ชั่นหลัก (setup, loop) ยังอยู่ตอนบนของโปรแกรมตลอดไม่ว่าจะมีฟังก์ชั่นรองกี่ฟังก์ชั่นก็ตามทำให้การตรวจสอบแก้ไขปรับปรุงในครั้งหลังทำได้สะดวก



(วางก่อนฟังก์ชั่นที่เรียกใช้งาน) (วางหลังฟังก์ชั่นที่เรียกใช้งาน)

รูปที่ 2-3 โครงสร้างการจัดวางฟังก์ชั่นรองในภาษาซี

วางก่อนฟังก์ชั่นที่เรียกใช้งาน

```
#define LED1 2
#define LED2 3
#define LED3 4
void pattern1(void)
{
  digitalWrite(LED1,HIGH);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,LOW);
}
void pattern2(void)
{
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,HIGH);
  digitalWrite(LED3,LOW);
}
void pattern3(void)
{
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,HIGH);
}
void setup()
{
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
}
void loop()
{
  pattern1();  delay(200);
  pattern2();  delay(200);
  pattern3();  delay(200);
}
```

วางหลังฟังก์ชั่นที่เรียกใช้งาน

```
#define LED1 2
#define LED2 3
#define LED3 4
void pattern1(void);
void pattern2(void);
void pattern3(void);
void setup()
{
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
}
void loop()
{
  pattern1();  delay(200);
  pattern2();  delay(200);
  pattern3();  delay(200);
}
void pattern1(void){
  digitalWrite(LED1,HIGH);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,LOW);
}
void pattern2(void){
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,HIGH);
  digitalWrite(LED3,LOW);
}
void pattern3(void){
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,HIGH);
}
```

ฟังก์ชั่นรองที่เขียนขึ้นใช้เองมีด้วยกัน 4 ประเภทคือ

- ฟังก์ชั่นที่ไม่รับค่าและไม่ส่งคืนค่า
- ฟังก์ชั่นที่รับค่าแต่ไม่ส่งคืนค่า
- ฟังก์ชั่นที่ไม่รับค่าแต่ส่งคืนค่า
- ฟังก์ชั่นที่รับค่าและส่งคืนค่า

ฟังก์ชั่นที่ไม่รับค่าและไม่ส่งคืนค่า

เป็นฟังก์ชั่นรองที่เขียนขึ้นโดยการรวมชุดฟังก์ชั่นเพื่อให้ทำงานอย่างหนึ่งอย่างใดเป็นอิสระจากฟังก์ชั่นหลักไม่มีการรับค่าใด ๆ จากฟังก์ชั่นหลักเพื่อนำใช้งานในตัวของฟังก์ชั่นรอง และไม่มีการส่งค่าใด ๆ กลับมายังฟังก์ชั่นหลักที่เป็นผู้เรียกใช้งาน ซึ่งข้อกำหนดในการประกาศใช้ของฟังก์ชั่นประเภทนี้มีดังนี้

1. ข้อกำหนดในการ**ไม่รับค่า** ทำโดยใส่ (void) หลังชื่อฟังก์ชั่น
2. ข้อกำหนดในการ**ไม่ส่งคืนค่า** ทำโดยใส่ void หน้าชื่อฟังก์ชั่น

ตัวอย่างโปรแกรมที่มีฟังก์ชั่นรองที่ไม่รับค่าและไม่ส่งคืนค่า void pattern1(void);

```
#define LED1 2
#define LED2 3
#define LED3 4
void pattern1(void);
void pattern2(void);
void pattern3(void);
void setup()
{
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
}
void loop()
{
  pattern1();  delay(200);
  pattern2();  delay(200);
  pattern3();  delay(200);
}
void pattern1(void)
{
  digitalWrite(LED1,HIGH);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,LOW);
}
void pattern2(void)
{
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,HIGH);
  digitalWrite(LED3,LOW);
}
void pattern3(void)
{
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,HIGH);
}
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 4, 5, 6 เป็นการประกาศรูปแบบของฟังก์ชั่นรองที่จะเขียนขึ้นใช้งาน (มี ; ปิดท้าย) และเป็นฟังก์ชั่นที่ไม่รับค่าและไม่ส่งคืนค่า (มี void หน้าฟังก์ชั่นและหลังฟังก์ชั่น)
- บรรทัดที่ 14, 15, 16 เป็นการเรียกใช้งานโดยการชื่อฟังก์ชั่นที่ต้องการตามด้วย ( ) ที่ภายในว่างเปล่าเนื่องจากไม่มีการส่งค่าเข้าในฟังก์ชั่น
- บรรทัดที่ 19-24 เป็นฟังก์ชั่นรองฟังก์ชั่นแรกที่เขียนขึ้นใช้งานโดยภายในบรรจุฟังก์ชั่นที่ต้องการให้ทำงานเมื่อมีการเรียกใช้
- บรรทัดที่ 25-30, บรรทัดที่ 25-30, บรรทัดที่ 31-36 เป็นฟังก์ชั่นรองถัด ๆ มาที่เขียนขึ้นใช้งาน

ฟังก์ชั่นที่รับค่าแต่ไม่ส่งคืนค่า

เป็นฟังก์ชั่นที่เขียนขึ้นเพื่อแบ่งย่อยการทำงานของฟังก์ชั่นหลัก โดยมีการส่งค่าจากฟังก์ชั่นหลักในขั้นตอนการเรียกใช้ฟังก์ชั่นรองเพื่อให้ฟังก์ชั่นรองที่เขียนขึ้นนำไปประมวลผลอย่างใดอย่างหนึ่งตามที่ผู้สร้างฟังก์ชั่นต้องการ ซึ่งข้อกำหนดในการประกาศใช้ของฟังก์ชั่นประเภทนี้มีดังนี้

1. ข้อกำหนดในการ**รับค่า** ทำโดยกำหนดชนิดตัวแปร พร้อมตัวแปรที่จะรับค่าท้ายชื่อฟังก์ชั่นรองที่สร้างขึ้น เช่น (byte data)
2. ข้อกำหนดในการ**ไม่ส่งคืนค่า** ทำโดยใส่ void หน้าชื่อฟังก์ชั่น

ตัวอย่างโปรแกรมที่มีฟังก์ชั่นรองที่รับค่าและไม่ส่งคืนค่า void send2port(byte data);

```
 1  #define LED1 2
 2  #define LED2 3
 3  #define LED3 4
 4  #define LED4 5
 5  char LED_pin[] = {LED1,LED2,LED3,LED4};
 6  void send2port(byte data);
 7  void setup()
 8  {
 9    Serial.begin(9600);
10    for(char i=0;i<4;i++)
11    {
12      pinMode(LED_pin[i],OUTPUT);
13    }
14  }
15  void loop()
16  {
17    send2port(0B1000);delay(200);
18    send2port(0B0100);delay(200);
19    send2port(0B0010);delay(200);
20    send2port(0B0001);delay(200);
21  }
22  void send2port(byte data)
23  {
24    digitalWrite(LED_pin[0],(data & 1 )? HIGH:LOW);
25    digitalWrite(LED_pin[1],(data & 2 )? HIGH:LOW);
26    digitalWrite(LED_pin[2],(data & 4 )? HIGH:LOW);
27    digitalWrite(LED_pin[3],(data & 8 )? HIGH:LOW);
28  }
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 6 เป็นการประกาศรูปแบบของฟังก์ชั่นรองที่จะเขียนขึ้นใช้งาน (มี ; ปิดท้าย) เป็นฟังก์ชั่นที่ไม่มีการส่งคืนค่า (มี void อยู่หน้าฟังก์ชั่น) แต่เป็นฟังก์ชั่นที่รับค่าเพียงอย่างเดียว โดยในวงเล็บหลังฟังก์ชั่นจะเป็นตัวแปรที่ใช้สำหรับรับค่าซึ่งจะต้องประกาศชนิดของตัวแปรพร้อมชื่อตัวแปรสำหรับรับค่าไว้ในวงเล็บท้ายฟังก์ชั่น
- บรรทัด 17-20 เป็นการเรียกใช้ฟังก์ชั่นรองโดยการเรียกชื่อฟังก์ชั่นรองพร้อมส่งค่าเข้าไปในฟังก์ชั่นโดยค่าที่ส่งไปจะใส่ในวงเล็บท้ายชื่อฟังก์ชั่นรองที่เรียกใช้ ค่าที่ส่งไปจะเข้าไปเก็บไว้ในตัวแปรที่ฟังก์ชั่นรองประกาศไว้ เช่น send2port(0B1000); เป็นการเรียกใช้ฟังก์ชั่น send2port โดยส่งค่าตัวเลข 1000 ซึ่งเป็นเลขฐานสองเข้าไปในฟังก์ชั่นรองนั้นด้วย ค่าตัวเลขดังกล่าวจะถูกเก็บไว้ในตัวแปร data ซึ่งฟังก์ชั่นรองได้เตรียมเอาไว้
- บรรทัดที่ 22-28 เป็นฟังก์ชั่นรองที่เขียนขึ้น ภายในฟังก์ชั่นมีการนำค่าที่ถูกส่งเข้ามาผ่านตัวแปร data นำมาใช้งาน (ใช้ฟังชั่น if-else แบบย่อรายละเอียดการใช้งานหน้าที่ 30)

ฟังก์ชั่นที่ไม่รับค่าแต่ส่งคืนค่า

เป็นฟังก์ชั่นที่เขียนขึ้นเพื่อแบ่งย่อยการทำงานของฟังก์ชั่นหลัก เพียงแต่ไม่ได้ส่งข้อมูลเข้าโปรแกรมเพื่อช่วยให้ทำการประมวลผลแต่มีการส่งข้อมูลกลับมายังฟังก์ชั่นหลัก เช่นฟังก์ชั่นตรวจสอบการกดสวิตช์ซึ่งฟังก์ชั่นนี้ไม่จำเป็นต้องรับข้อมูลใดมาประมวลผลมีเพียงตรวจสอบว่ามีการกดสวิตช์หรือไม่แล้วส่งค่าการกดสวิตช์กลับไปยังฟังก์ชั่นหลักเพื่อนำไปใช้งานต่อไป โดยค่าที่ส่งกลับผู้เขียนโปรแกรมจะเป็นผู้กำหนดขึ้นมาเอง เช่น เมื่อสวิตช์ 1 ถูกกดให้ส่งค่า 1 กลับและเมื่อสวิตช์ 2 ถูกกดให้ส่งค่า 2 หากไม่มีการกดใด ๆ ให้ส่งกลับค่า 0 เป็นต้น

การเขียนฟังก์ชั่นรองลักษณะนี้มีข้อกำหนดในการประกาศใช้ของฟังก์ชั่นดังนี้

1. ข้อกำหนดในการ**ไม่รับค่า** ทำโดยใส่ (void) หลังชื่อฟังก์ชั่น
2. ข้อกำหนดในการ**ส่งคืนค่า** ทำโดยใส่กำหนดชนิดของข้อมูลที่จะส่งคืนหน้าชื่อฟังก์ชั่นรองที่สร้างขึ้น

ตัวอย่างโปรแกรมที่มีฟังก์ชั่นรองที่ไม่รับค่าแต่ส่งคืนค่า byte readSW(void);

```
 1  #define LED 2
 2  #define SW1 3
 3  #define SW2 4
 4  byte readSW(void);
 5  void setup()
 6  {
 7    pinMode(LED,OUTPUT);
 8    pinMode(SW1,INPUT_PULLUP);
 9    pinMode(SW2,INPUT_PULLUP);
10  }
```

```
11  void loop()
12  {
13    byte x=readSW();
14    if(x==1)
15      digitalWrite(LED,HIGH);
16    else if(x==2)
17      digitalWrite(LED,LOW);
18  }
19  byte readSW(void)
20  {
21    byte Status=0;
22    if(digitalRead(SW1)==LOW)
23      Status=1;
24    else if(digitalRead(SW2)==LOW)
25      Status=2;
26    return Status;
27  }
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 4 เป็นการประกาศรูปแบบของฟังก์ชั่นรอง byte readSW(void); ซึ่งเป็นฟังก์ชั่นที่มีการส่งค่ากลับคืนฟังก์ชั่นหลัก โดยค่าที่ส่งกลับมีขนาดเป็นค่าของตัวแปร byte
- บรรทัดที่ 13 เป็นการเป็นการเรียกใช้ฟังก์ชั่นรอง byte x=readSW(); เมื่อฟังก์ชั่นรองทำงานเสร็จสิ้นจะส่งค่าเข้ามายังตัวแปร x ซึ่งเป็นตัวแปรได้ประกาศชนิดที่มีขนาดที่เพียงพอที่สามารถรับข้อมูลที่ส่งกลับมาจากฟังก์ชั่นรองนี้ได้
- บรรทัด 19-27 เป็นฟังก์ชั่นรองที่เขียนขึ้นเพื่อใช้งานที่มีการส่งค่าคืนไปยังฟังก์ชั่นหลัก
- บรรทัดที่ 21 การประกาศตัวแปรที่มีชนิดเดียวกันกับค่าที่ฟังก์ชั่นรองส่งออก
- บรรทัดที่ 26 ฟังก์ชั่น return เป็นฟังก์ชั่นที่ใช้สำหรับส่งค่าออกจากฟังก์ชั่นรอง โดยค่าที่ส่งออกจะอยู่ในตัวแปรท้ายฟังก์ชั่น return ซึ่งมีขนาดเดียวกันกับชนิดของค่าที่ประกาศของฟังก์ชั่น (หน้าชื่อฟังก์ชั่นรอง)

ฟังก์ชั่นที่รับค่าและส่งคืนค่า

เป็นฟังก์ชั่นที่เขียนขึ้นเพื่อแบ่งย่อยการทำงานของฟังก์ชั่นหลักโดยมีการส่งค่าจากฟังก์ชั่นหลักเพื่อให้ฟังก์ชั่นรองที่เขียนขึ้นนำไปประมวลค่าอย่างใดอย่างหนึ่งและส่งค่ากลับมายังฟังก์ชั่นหลัก ซึ่งฟังก์ชั่นรองชนิดนี้มักใช้บ่อยเมื่อต้องการสร้างกลุ่มฟังก์ชั่นที่ต้องการคำนวณสิ่งใดสิ่งหนึ่งโดยมีข้อมูลที่ใช้ในการคำนวณนั้น ๆ ด้วย และเมื่อคำนวณเสร็จสิ้นฟังก์ชั่นหลักมีความต้องการผลการคำนวณนั้นด้วย เช่น ฟังก์ชั่นแปลงค่าจากการอ่านค่าจากเทอร์มิสเตอร์ NTC และต้องการผลเป็นอุณหภูมิ ซึ่งข้อกำหนดในการประกาศใช้ของฟังก์ชั่นประเภทนี้มีดังนี้

1. ข้อกำหนดในการ**รับค่า** ทำโดยกำหนดชนิดตัวแปร พร้อมตัวแปรที่จะรับค่าท้ายชื่อฟังก์ชั่นรองที่สร้างขึ้น เช่น (int x)
2. ข้อกำหนดในการ**ส่งคืนค่า** ทำโดยใส่กำหนดชนิดของข้อมูลที่จะส่งคืนหน้าชื่อฟังก์ชั่นรอง

ตัวอย่างโปรแกรมที่มีฟังก์ชั่นรองที่รับค่าและส่งคืนค่า double Thermistor(int RawADC);

```
 1  #define NTC A5
 2  double Thermistor(int RawADC);
 3  void setup()
 4  {
 5    Serial.begin(9600);
 6  }
 7  void loop()
 8  {
 9    float Temp=Thermistor(analogRead(NTC));
10    Serial.print("Temperature is : ");
      Serial.print(Temp);
11    Serial.println(" 'C");
12    delay(1000);
13  }
14  double Thermistor(int RawADC)
15  {
16    double Cal;
17    Cal = log(10000.0/((1024.0/RawADC-1)));
18    Cal = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Cal * Cal ))* Cal );
19    Cal = Cal - 273.15;            // Convert Kelvin to Celcius
20    return Cal;
21  }
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 2 เป็นการประกาศรูปแบบของฟังก์ชั่นรอง double Thermistor(int RawADC); ที่จะเขียนขึ้นใช้งาน (มี ; ปิดท้าย) เป็นฟังก์ชั่นที่มีการรับค่าและส่งค่ากลับคืนฟังก์ชั่นหลัก โดยค่าที่รับจะถูกเก็บไว้ในตัวแปรชื่อ RawADC ถูกกำหนดให้เป็นชนิด integer และส่งคืนค่ากลับมีขนาดเป็นค่าของตัวแปร double
- บรรทัดที่ 9 เป็นการเรียกใช้ฟังก์ชั่นรอง float Temp=Thermistor(analogRead(NTC)); โดยส่งค่าเข้าฟังก์ชั่นรองในวงเล็บหลังชื่อฟังก์ชั่น และประกาศตัวแปรไว้รับค่า (Temp) ที่ได้จากการส่งกลับจากฟังก์ชั่นรอง จากตัวอย่างจะเห็นว่าใช้ตัวแปรที่ใช้รับค่ามีขนาดเล็กกว่าค่าที่ส่งกลับ ดังนั้นจะมีข้อมูลหายไปบางส่วน แต่ถ้าหากผู้เขียนโปรแกรมยอมรับการสูญหายของข้อมูลนั้นได้โดยผลการทำงานของโปรแกรมยังเท่าเดิมก็สามารถทำได้
- บรรทัดที่ 14-21 เป็นฟังก์ชั่นรองที่เขียนขึ้นมาเพื่อประมวลผลตามต้องการที่มีการรับค่าเข้ามาคำนวณและส่งค่ากลับคืนยังฟังก์ชั่นหลัก

2.4 ขอบเขตของตัวแปร

ขอบเขตของตัวแปรหมายถึงบริเวณหรือตำแหน่งหรือพิกัดของตัวแปรที่ฟังก์ชั่นสามารถเรียกใช้งานได้ ทั้งนี้ขึ้นอยู่กับตัวแปรตัวนั้น ๆ ว่าประกาศไว้ที่จุดใด โดยขอบเขตของตัวแปรมีด้วยกัน 2 แบบ

1. **ตัวแปรประเภทโกลบอล (global)** เป็นตัวแปรที่ประกาศนอกฟังก์ชั่นซึ่งจะประกาศไว้ในส่วนของหัวโปรแกรม ตัวแปรเหล่านี้สามารถถูกนำไปใช้งานได้ทุกฟังก์ชั่น หรืออาจกล่าวได้ว่าทุกฟังก์ชั่นสามารถมองเห็นตัวแปรประเภทนี้

2. **ตัวแปรประเภทโลคอล (local)** เป็นตัวแปรที่ประกาศภายในฟังก์ชั่นซึ่งสามารถใช้งานได้เฉพาะภายในฟังก์ชั่นที่ประกาศเท่านั้นนอกฟังก์ชั่นไม่สามารถใช้งานได้ หรืออาจกล่าวได้ว่าตัวแปรโลคอลมองเห็นเฉพาะภายในเท่านั้น

ตัวอย่างการประกาศตัวแปรในขอบเขตที่ต่างกัน

```
 1  byte LED1=2;
 2  byte LED2=3;
 3  void setup()
 4  {
 5    pinMode(LED1,OUTPUT);
 6    pinMode(LED2,OUTPUT);
 7  }
 8  void loop()
 9  {
10    int i;
11    for(i=0;i<10;i++)
12    {
13      digitalWrite(LED1,HIGH);delay(100);
14      digitalWrite(LED1,LOW);delay(100);
15    }
16    for(i=0;i<10;i++)
17    {
18      digitalWrite(LED2,HIGH);delay(100);
19      digitalWrite(LED2,LOW);delay(100);
20    }
21  }
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 1,2 เป็นการประกาศตัวแปร LED1, LED2 ซึ่งประกาศอยู่นอกฟังก์ชั่นเป็นการประกาศตัวแปรในตำแหน่งโกลบอล นั่นหมายความว่าทุกฟังก์ชั่นจะสามารถใช้ (มองเห็น) ตัวแปรนี้ได้ สามารถนำไปใช้งานได้ เช่นในฟังก์ชั่น setup ได้นำไปกำหนดโหมดการทำงานในฟังก์ชั่น pinMode และในฟังก์ชั่น loop ได้นำไปใช้ในฟังก์ชั่น digitalWrite
- บรรทัดที่ 10 เป็นการประกาศตัวแปร i ซึ่งจะนำไปใช้นับรอบในคำสั่ง forโดยตัวแปรนี้จะมองเห็นและใช้งานได้เฉพาะภายในฟังก์ชั่น loop เท่านั้น นอกฟังก์ชั่น (เช่นฟังก์ชั่น setup ในตัวอย่าง) ไม่สามารถใช้งานตัวแปรนี้ได้ หรือกล่าวได้ว่ามองไม่เห็นตัวแปร i นี้

2.5 การประกาศตัวแปรแบบสตรัคเจอร์ (Structure) และยูเนียน (Union)

การใช้งานตัวแปรที่มีจำนวนตัวแปรมาก ๆ อาจเกิดการสับสนในการใช้งานได้ ภาษาซีสามารถจัดรวมกลุ่มตัวแปรเพื่อให้เกิดความสะดวกต่อการเรียกใช้งานได้ ซึ่งตัวแปรที่นำมารวมกลุ่มนั้นสามารถรวมกลุ่มตัวแปรที่เป็นชนิดแตกต่างกันได้ โดยมีวิธีการประกาศด้วยกัน 2 แบบคือ

1. แบบสตัคเจอร์ (Structure)
2. แบบยูเนียน (Union)

โดยในแต่ละแบบมีคุณสมบัติในการจัดการที่แตกต่างกัน

การประกาศตัวแปรแบบสตัคเจอร์ (Structure)

เป็นการประกาศตัวแปรที่สามารถรวมกลุ่มของข้อมูลได้หลายชนิดไม่ว่าจะเป็นตัวเลขจำนวนเต็ม ตัวเลขที่เป็นทศนิยมหรือเป็นตัวอักษรก็ตาม โดยสามารถใช้งานได้ในเวลาเดียวกันเนื่องจากตัวแปรย่อย ๆ ภายในสตัคเจอร์ได้ถูกแยกพื้นที่หน่วยความจำสำหรับการเก็บข้อมูล วิธีการประกาศมีรูปแบบดังนี้

การประกาศ

```
struct ชื่อตัวแปรสตัคเจอร์
{
        ชนิดตัวแปร ชื่อตัวแปรตัวที่ 1;
        ชนิดตัวแปร ชื่อตัวแปรตัวที่ 2;
                ---
};
```

การใช้งาน

ชื่อตัวแปรสตัคเจอร์ ชื่อตัวแปรใหม่ที่ต้องการตั้งชื่อ={ค่าเริ่มต้นของตัวแปรตัวที่ 1, 2, …};

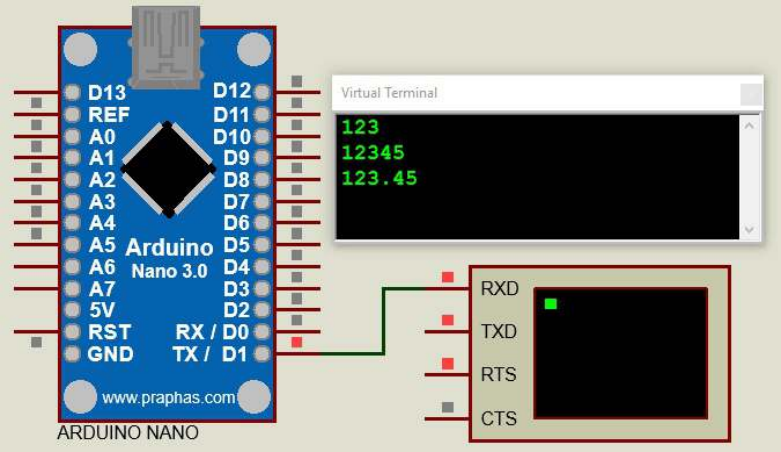
ตัวอย่างเช่น

```
 1  struct VALUE
 2  {
 3    byte a;
 4    int b;
 5    float c;
 6  };
 7  VALUE data={123,12345,123.45};
 8  void setup()
 9  {
10    Serial.begin(9600);
11    Serial.println(data.a);
12    Serial.println(data.b);
13    Serial.println(data.c);
14  }
15  void loop()
16  {
17  }
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 1-6 เป็นการประกาศตัวแปรแบบสตัคเจอร์โดยตั้งชื่อว่า VALUE มีตัวแปรภายใน 3 ตัว ที่เก็บข้อมูลแตกต่างกัน
- บรรทัดที่ 7 เป็นการกำหนดค่าเริ่มต้นโดยสามารถเปลี่ยนชื่อเรียกใหม่ได้ ในที่นี้เปลี่ยนชื่อจาก VALUE เป็นชื่อว่า data และสามารถกำหนดค่าเริ่มต้นเข้าไปในทุกตัวแปรภายในได้ในขั้นตอนนี้ จากตัวอย่างกำหนดตัวแปร data.a=123 ตัวแปร data.b=12345 และตัวแปร data.c=123.45

ผลการรันเป็นดังรูป



รูปที่ 2-4 ผลการรันโปรแกรมที่ใช้งานตัวแปรแบบสตัคเจอร์

การประกาศตัวแปรแบบยูเนียน (Union)

เป็นการประกาศตัวแปรที่สามารถรวมกลุ่มของข้อมูลได้หลายชนิดเช่นเดียวกับแบบสตัคเจอร์ แต่มีความแตกต่างตรงที่แบบยูเนียนใช้หน่วยความจำเก็บข้อมูลในตำแหน่งเดียวกันในทุกตัวแปรดังนั้นเวลาใช้งานจึงไม่สามารถใช้พร้อมกันได้ เนื่องจากตัวแปรที่กำหนดค่าตัวหลังสุดจะเป็นตัวที่ได้ใช้งานหน่วยความจำในตำแหน่งนั้น ๆ ทำให้ข้อมูลในตัวแปรแรก ๆ ถูกแทนที่ไปด้วยค่าในตัวแปรล่าสุดแทน รูปแบบการประกาศดังนี้

การประกาศ

union ชื่อตัวแปรยูเนียน
{
ชนิดตัวแปร ชื่อตัวแปรตัวที่ 1;
ชนิดตัวแปร ชื่อตัวแปรตัวที่ 2;
--- 	
};

การใช้งาน
ชื่อตัวแปรยูเนียน ชื่อตัวแปรใหม่ที่ต้องการตั้งชื่อ;

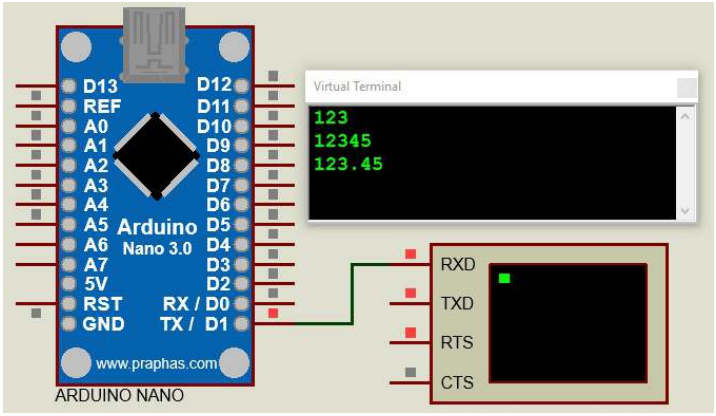
ตัวอย่างเช่น

```
1  union VALUE
2  {
3    byte a;
4    int b;
5    float c;
6  };
7  VALUE data;
8  void setup()
9  {
10   Serial.begin(9600);
11   data.a=123;  Serial.println(data.a);
12   data.b=12345;  Serial.println(data.b);
13   data.c=123.45;  Serial.println(data.c);
14 }
15 void loop()
16 {
17 }
```

รายละเอียดโค้ดโปรแกรม

- บรรทัดที่ 1-6 เป็นการประกาศตัวแปรแบบยูเนียนโดยตั้งชื่อว่า VALUE มีตัวแปรภายใน 3 ตัวที่เก็บข้อมูลแตกต่างกัน
- บรรทัดที่ 7 เป็นการกำหนดชื่อเรียกใหม่ในที่นี้เปลี่ยนชื่อจาก VALUE เป็นชื่อว่า data แต่ไม่สามารถกำหนดค่าเริ่มต้นแบบตัวแปรสตรัคเจอร์ได้
- บรรทัดที่ 11 เป็นการกำหนดค่าให้กับตัวแปรตัวแรก data.a พร้อมนำไปใช้งานทันที โดยตัวอย่างดังกล่าวเป็นการนำส่งข้อมูลออกทางพอร์ตอนุกรมเพื่อแสดงผลหน้าจอคอมพิวเตอร์
- บรรทัดที่ 12 เป็นการกำหนดค่าให้กับตัวแปรตัวแรก data.b ค่าของตัวแปรนี้จะไปทับข้อมูลของ data.a ซึ่งตอนนี้ไม่ได้ใช้งานแล้ว
- บรรทัดที่ 13 เป็นการกำหนดค่าให้กับตัวแปรตัวแรก data.c ค่าตัวแปรนี้จะไปทับข้อมูลของตัวแปร data.b เนื่องจากใช้ตำแหน่งของหน่วยความจำเดียวกันในการเก็บข้อมูล

ผลการรันเป็นดังรูป



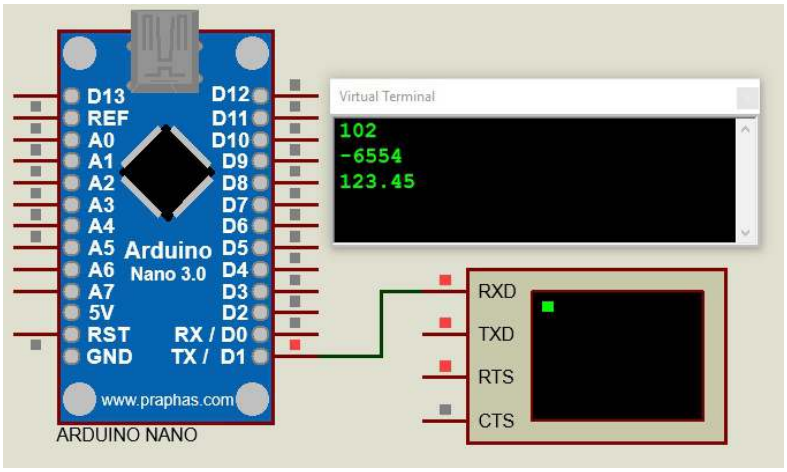
รูปที่ 2-5 ผลการรันโปรแกรมที่ใช้งานตัวแปรแบบยูเนียน

หากกำหนดค่าตัวแปรต่อเนื่องกัน ตัวแปรตัวสุดท้ายจะไปทับข้อมูลตัวแปรตัวแรก ๆ ของตัวแปรยูเนียน

ตัวอย่างเช่น

```
union VALUE
{
  byte a;
  int b;
  float c;
};
VALUE data;
void setup()
{
  data.a=123;
  data.b=12345;
  data.c=123.45;
  Serial.begin(9600);
  Serial.println(data.a);
  Serial.println(data.b);
  Serial.println(data.c);
}
void loop()
{
}
```

ผลการรันเป็นดังรูป



รูปที่ 2-6 ผลการรันโปรแกรมที่ใช้งานตัวแปรแบบยูเนียนที่กำหนดค่าตัวแปรพร้อมกัน