โปรแกรมภาษา C บน Arduino

The Building Blocks of All Programming Languages

All programming languages, from Ada to ZPL, are built from four basic elements:

1) Expressions
2) Statements
3) Statement blocks
4) Function blocks

Expressions

An expression is created by combining operands and operators.

a + b

m – 3000

g < d

A + B + C

Statements

A statement is a complete C instruction for the computer. All C statements end with a semicolon (;). The following are examples of C statements:

i = 50;

a = b + c;

m = d / 2;.

Operator Precedence

Suppose you have the following statement comprised of several expressions:

ตัวอย่างเช่น

j = 5 + k * 2;

ถ้า k = 3 j = 11 หรือ 16 ?

Table 2-1. *Operator Precedence*

Precedence level	Operator
1	* (multiplication), /, %
2	+, -

% Modulus Operator and remainder of after an integer division.

เช่นถ้า A = 10 B = 20 ถ้า B % A = 0

Statement Blocks

A statement block consists of one or more statements grouped together so they are viewed by the compiler as though they are a single statement. For example, suppose you are an apartment manager, and if there is 4 or more inches of snow on the ground, then you need to shovel the sidewalk. You might express this as (the >= operator is read as "greater than or equal to"):

```
if (snow >= 4) {
        PutOnSnowRemovalStuff();
        GetSnowShovel();
        ShovelSidewalk();
} else {
        GoBackToBed();
}
```

Statement blocks start with an opening brace character { and end with a closing brace character }.

Function Blocks

A function block is a block of code that is designed to accomplish a single task. Although you may not be aware of it, you actually used a function block in the previous section. That is, PutOnSnowRemovalStuff() is

a function that is designed to have you put on your coat. The actual code might look like:

```
void PutOnSnowRemovalStuff(void) {
if (NotDressed) {
PutOnClothes();
PutOnShoes();
}
GoToCloset();
PutOnBoots();
PutOnCoat();
PutOnGloves();
PutOnHat();
}
```

The Five Program Steps

1) Initialization Step
2) Input Step
3) Process Step
4) Output Step
5) Termination Step

Initialization Step

The purpose of the Initialization Step is to establish the environment in which the program will run.

Input Step

Almost every computer program has a task that is designed to take some existing state of information, process it in some way, and show the new state of that information. If you are writing a fire alarm system, then you take the information provided by the fire sensors, interpret their current state and, if there is a fire, do something about it.

Process Step

Continuing with our fire alarm program, once the input from the sensors is received, some body of code must be responsible for determining whether the sensors are detecting a fire. In other words, the voltage (i.e., temperature) must be read (input) and then interpreted (i.e., the data processed) to determine the current state of the sensors. In a desktop application, perhaps the data input is the price and quantity of some item purchased by a customer. The Process Step may perform the task of determining the total cost to the consumer.

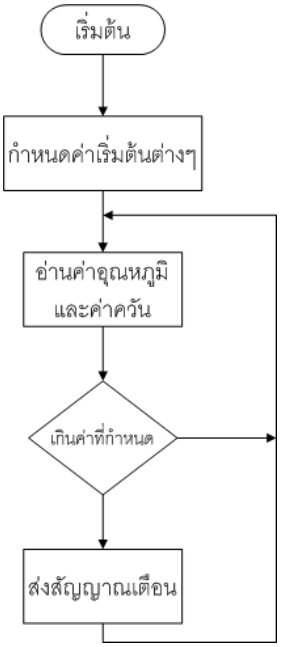
Output Step

After the Process Step has finished its work, the new value is typically output on some display device.

Termination Step

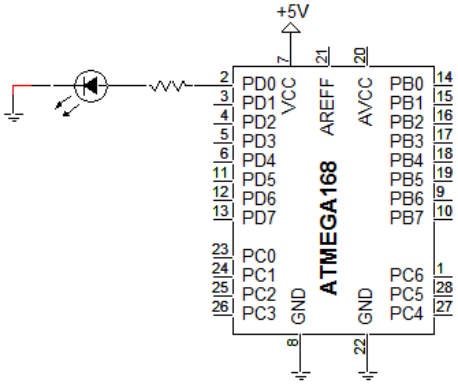
The Termination Step has the responsibility of "cleaning up" after the program is finished performing its task. In desktop applications, it is common for the Termination Step to perform the Initialization Step "in reverse." That is, if the program keeps track of the most recent data files that were used, then the termination Step must update that list of files. If the Initialization Step opens a database or printer connection, then the Termination Step should close that connection down so unused resources are available to the system.

การทำงานโดยทั่วไปของระบบควบคุม



ภาษา C ของ AVR

```
#define F_CPU 1000000UL  /* 1 MHz CPU clock */
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
int main(void)
{
// initialize the PORT D as an output
    DDRD = 0x01;
   while(1)
   {
         PORTD = 0x01; //turn the LED on
         _delay_ms(500);          //wait for 500 ms
         PORTD = 0x00; //turn the LED off
         _delay_ms(500);          //wait for 500 ms
   }
}
```



ภาษา C ของ Arduino

```
int led = 0;       //Digital Pin 0 = D0
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
         pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
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
}
```

เปรียบเทียบ

ภาษา C ของ AVR

```
#define F_CPU 1000000UL  /* 1 MHz CPU
    clock */
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
int main(void)
{
// initialize the PORT D as an output
    DDRD = 0x01;
   while(1)
   {
    PORTD = 0x01;    //turn the LED on
    _delay_ms(500);  //wait for 500 ms
    PORTD = 0x00;    //turn the LED off
    _delay_ms(500);  //wait for 500 ms
   }
}
```

ภาษา C ของ Arduino

```
int led = 0;   //Digital Pin 0 = D0
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off
  delay(1000);               // wait for a second
}
```

เขียนอีกแบบ

```
#define ledpin 13
#define led_on digitalWrite(ledpin, HIGH)
#define led_off digitalWrite(ledpin, LOW)
void setup() {
    pinMode(ledpin, OUTPUT); // initialize the digital pin as an output.
}
void loop() {
    led_on; // set the LED on
    delay(200); // wait for a second
    led_off; // set the LED off
    delay(200); // wait for a second
}
```

Arduino C Data Types

Type	Byte length	Range of values
boolean	1	Limited to logic true and false
char	1	Range: –128 to +127
unsigned char	1	Range: 0 to 255
byte	1	Range: 0 to 255
int	2	Range: –32,768 to 32,767
unsigned int	2	Range: 0 to 65,535
word	2	Range: 0 to 65,535
long	4	Range: –2,147,483,648 to 2,147,483,647
unsigned long	4	Range: 0 to 4,294,967,295
float	4	Range: –3.4028235E+38 to 3.4028235E+38
double	4	Range: –3.4028235E+38 to 3.4028235E+38
string	?	A null ('\0') terminated reference type data build from a character array
String	?	An reference data type object
array	?	A sequence of a value type that is referenced by a single variable name
void	0	A descriptor used with functions as a return type when the function does not return a value.

Keywords in C

- The keywords are reserved for the compiler's use, you cannot use them for your own variable or function names.
- Each of the types shown in Table (i.e., boolean, char, int, etc.) are keywords in C. A keyword is any word that has special meaning to the C compiler.
- If you do, the compiler will flag it as an error. If the compiler didn't flag such errors, then the compiler would be confused as to which use of the keywords to use in any given situation.

Variable Names in C

There are three general rules for naming variables or functions in C:

Valid variable names may contain:

1) Characters a through z and A through Z
2) The underscore character (_)
3) Digit characters 0 through 9, provided they are not used as the first character in the name.

Valid variable names might include:

jane Jane ohm ampere volt money day1 Week50_system XfXf

The following would not be valid names:

^carat 4July -negative @URL %percent not-Good This&That what?

Built-In String Functions

Table 3-3. Built-In String Functions

Function	Purpose
`String()`	Define a `String` object.
`charAt()`	Access a character at a specified index.
`compareTo()`	Compare two `Strings`.
`concat()`	Append one `String` to another `String`.
`endsWith()`	Get the last character in the `String`.
`equals()`	Compare two `Strings`.
`equalsIgnoreCase()`	Compare two `Strings`, but ignore case differences.
`getBytes()`	Copies a `String` into a byte array.
`indexOf()`	Get the index of a specified character.
`lastIndexOf()`	Get the index of the last occurrence of a specified character.
`length()`	The number of characters in the `String`, excluding the null character.
`replace()`	Replace one given character with another given character.
`setCharAt()`	Change the character at a specific index.
`startsWith()`	Does one `String` begin with a specified sequence of characters?
`substring()`	Find a substring within a `String`.
`toCharArray()`	Change from `String` to character array.
`toLowerCase()`	Change all characters to lower case.
`toUpperCase()`	Change all characters to upper case.
`trim()`	Remove all whitespace characters from a `String`.

The void Data Type

The void data type really isn't a data type at all.

It is used with functions to show that a function does not return a value. For example,

```
void setup() {
// the setup code body
}
void loop() {
// the loop code body
}
```

The use of void here means that no data are returned from either of these two functions.

Another use of void is to say that no information is passed to the function. In other words, you could write the two functions as:

```
void setup(void) {
// the setup code body
}
void loop(void) {
// the loop code body
}
```

The array Data Type

Virtually all of the basic data types support arrays of those types. The following statements show some other array definitions:

```
int myData[15];
long yourWorkDay[7];
float temp[200];
```

Decision Making in C

Table 4-1. Relational Operators

Operator	Interpretation
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

ตัวอย่าง

```
5 > 4          // Logic true
5 < 4          // logic false
5 == 4         // logic false
5 != 4         // logic true
```

If a = 5 and b = 4, then:

```
a > b          // Logic true
a < b          // logic false
a == b         // logic false
a != b         // logic true
```

Decision Statement

- If Statement
- Switch Statement

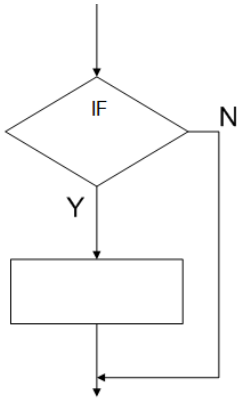
The syntax for an if statement is:

if (expression1 is logic true) {

// execute this if statement block if true

}

// statements following the if statement block



The if-else Statement

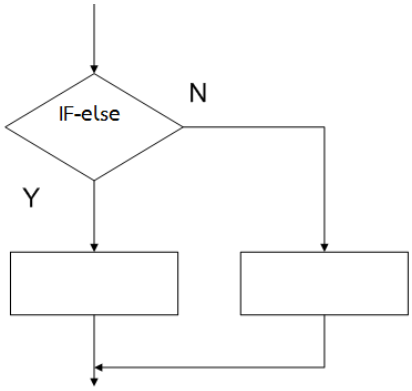
if (expression evaluates to logic true) {

// perform this statement block if logic true

} else {

// perform this statement block otherwise

}



Cascading if Statements

```
int myDay;
// Some code that determines what day it is...
if (myDay == 1) {
doSundayStuff();
}
if (myDay == 2) {
doMondayStuff();
}
if (myDay ==3) {
doTuesdayStuff();
}
if (myDay == 4) {
doWednesdayStuff();
}
if (myDay == 5) {
doThursdayStuff();
}
if (myDay == 6) {
doFridayStuff();
}
if (myDay == 7) {
doSaturdayStuff();
}
```

The switch Statement

The switch statement has the following syntax:

```
switch (expression1) { // opening brace for switch statement block
case 1:
          // statements to execute when expression1 is 1
          break;
case 2:
          // statements to execute when expression1 is 2
          break;
case 3:
          // statements to execute when expression1 is 3
          break;
// more case statements as needed
default:
          // statements to execute if expression1 doesn't have a "case value"
          break;
} // close brace for switch statement block
/
/ This is the next statement after the switch
```

ตัวอย่าง

```
switch (inByte) {
      case 'a':
          digitalWrite(2, HIGH);
          break;
      case 'b':
          digitalWrite(3, HIGH);
          break;
      case 'c':
          digitalWrite(4, HIGH);
          break;
      case 'd':
          digitalWrite(5, HIGH);
          break;
       case 'e':
          digitalWrite(6, HIGH);
          break;
       default:
          for (int thisPin = 2; thisPin < 7; thisPin++) {
            digitalWrite(thisPin, LOW);
          }
          break;
}
```

เขียนอีกสไตล์

```
switch (inByte) {
     case 'a':  digitalWrite(2, HIGH);  break;
     case 'b':  digitalWrite(3, HIGH);  break;
     case 'c':  digitalWrite(4, HIGH);  break;
     case 'd':  digitalWrite(5, HIGH);  break;
     case 'e':  digitalWrite(6, HIGH);  break;
     default:
       for (int thisPin = 2; thisPin < 7; thisPin++) {
         digitalWrite(thisPin, LOW);
       }
       break;
}
```

Precedence Operator

Table 4-2. Precedence of Operators

Level	Operators
1	() [] → . (dot)
2	! ~ ++ - - + (unary) - (unary) * (indirection) & (address of) (cast) sizeof
3	* (multiplication) / %
4	+ (binary) - (binary)
5	<< >>
6	< <= > >=
7	== !=
8	& (bitwise AND)
9	^
10	|
11	&&
12	||
13	?:
14	= += - = *= /= %= &= ^= |= <<= >>=

The Increment and Decrement Operators

counter = counter + 1;

Two Types of Increment Operator (++) There are two flavors forthe increment operator:

1) pre-increment

++counter;

The counter is incremented and then used in the expression in which it happens to appear.

2) post-increment

counter++;

The counter is fetched and used in the expression and then incremented

```
int c = 5;
```

```
int k;
k = ++c; // pre-increment, k == 6, c == 6
k = c++; // post-increment, k == 5, c == 6
```

Two Flavors of the Decrement Operator (--)

The decrement operator (--) is similar to the increment operator but is used to decrease the rvalue of a variable by 1. That is,

counter = counter – 1;

could be written

counter--;

post-decrement operation.

k = --c; // If C = 5 then C = 4 and k = 4

The post-decrement operator:

k = c--; // If C = 5 then C = 4 and k = 5

Program Loops in C

1) For loop
2) The while statement
3) A do-while statement and its differences
4) Infinite loops
5) The break and continue keywords

For Loop

The general syntax structure of a for loop is as follows:

```
for (expression1; expression2; expression3) {
// for loop statement body
}
// the first statement following the for loop structure
```

```
for(int x = 2; x < 10; x ++){
        println(x);
}
```

1. เริ่มต้นให้ X = 2
2. พิมพ์ค่า X
3. ตรวจสอบว่า X ยังน้อยกว่า 10 หรือไม่ ถ้าใช้ เพิ่มค่า X แล้วกลับไปข้อ 2 ใหม่ ถ้าไม่ใช่ให้ออกจาก Loop ทำคำสั่งถัดไป

```
void loop() {
 for (int thisPin = 2; thisPin < 8; thisPin++) {
    digitalWrite(thisPin, HIGH); // turn the pin on:
    delay(300);
    digitalWrite(thisPin, LOW); // turn the pin off:
  }
}
```

1. เริ่มต้นให้ thispin = 2
2. ทำให้ขา thispin เป็นลอจิก 1
3. หน่วงเวลา 300 ms
4. ทำให้ขา thispin เป็นลอจิก 0
5.ตรวจสอบว่า thispin ยังน้อยกว่า 8 หรือไม่ ถ้าใช้ เพิ่มค่า thispin แล้วกลับไปข้อ 2 ใหม่ ถ้าไม่ใช่ให้ออกจาก Loop ทำคำสั่งถัดไป

The while statement

The syntax of the while loop is:

```
while (expression2) {
  // Statements in the loop body
} // End of while statement bloc
```

ถ้าเงื่อนไข expression เป็นจริงวนทำ Statements in the loop body อยู่ใน loop

ตัวอย่าง

```
k = 0;                  // กำหนดค่าเริ่มต้นให้
while (k < 1000) {      //ตรวจสอบถ้า k น้อยกว่า 1000 ให้ทำคำสั่งใน loop ถ้าไม่ใช่ ให้ออกนอก loop
        println(k);     //พิมพ์ค่า k แล้วขึ้นบรรทัดใหม่
        k++;            //เพิ่มค่า k ไป 1  แล้วกลับไปตรวจสอบค่า k
}
```

A do-while statement and its differences

The third type of loop structure is the do-while loop. The syntax is:

```
do {
   // Loop body statements
} while (expression2);
```

ลูปแบบนี้จะทำคำสั่งในลูปก่อน แล้วจึงมาตรวจสอบ expression 2 ถ้าเป็นจริงก็ย้อนกลับไปทำคำสั่งในลูปใหม่ ถ้าไม่จริง ก็ออกจากลูป ทำคำสั่งถัดไป

```
do {
        delay(50);              // หน่วงเวลา 50 mS
        x = readSensors();      // อ่านค่า sensor สมมุติว่าในโปรแกรมมี ฟังก์ชั่น  readSensors()
} while (x < 100);              //ตรวจสอบค่าที่อ่านได้ ถ้าน้อยกว่า 100 กลับ/ทำคำสั่งหน่วงเวลาและอ่าน sensor
y = x-100;                       //ใหม่ ถ้าเท่ากับหรือมากกว่า 100 ให้ออกนอกลูปแล้วลงไปทำคำสั่ง  y = x-100;
```

Infinite loops

```
while (true) {
        ...
  statements
        ..
}
```

```
while (1) {
        ...
  statements
        ..
}
```

เลข 1 ในวงเล็บมีค่าเท่ากับ True ดังนั้นเงื่อนไขเป็นจริงตลอด คำสั่งจึงบังคับให้ทำอยู่ในลูปตลอดไป

The break and continue keywords

break

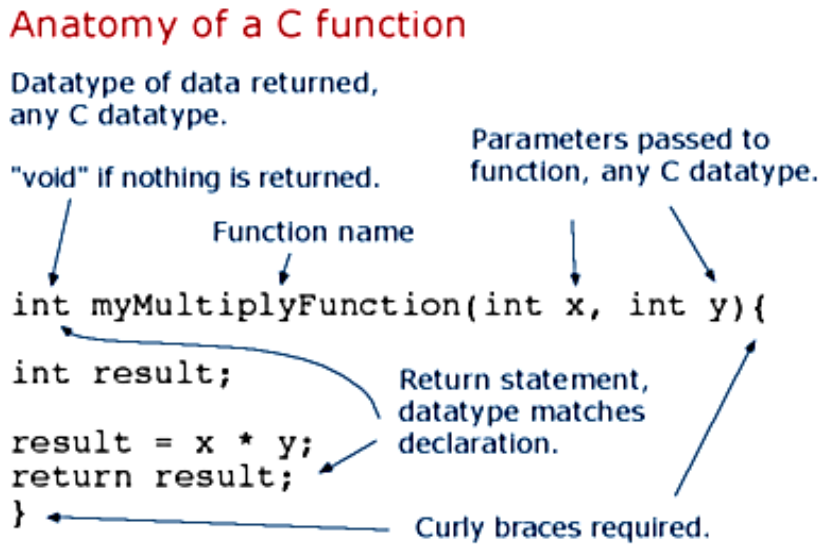
```
x = 0;                                  //กำหนดค่าให้ x = 0
while (x < 100){
        x = readSensors();
        If (x>50 && x<60) {
                Break;
        }
}
```

continue

```
For (i = 0; x < 8; i++)                 //ให้ i มีค่าตั้งแต่ 0 ถึง 7 (น้อยกว่า 8)
{
        if( i % 2 ==0) continune;       // ถ้า i หารด้วย 2 เศษเท่ากับ 0 ให้ข้ามคำสั่งไปถึงจุดวนลูป
        digitalWrite(i,HIGH);           //ให้ขาดิจิตอลเอาท์พุทที่ i เป็น 1
        delay(500);                     //หน่วงเวลา 500 ms
         digitalWrite(i,LOW);           //ให้ขาดิจิตอลเอาท์พุทที่ i เป็น 0
        delay(500);                     //หน่วงเวลา 500 ms
}                                       //จุดสิ้นสุดลูป เพิ่มค่า i แล้วกลับไปเริ่มต้นใหม่
```

การทำงานของตัวอย่างนี้เป็นการทำให้ ขาดิจิตอลเอาท์พุทที่ เป็นเลขคี่ เป็น 1 แล้วเป็น 0

Functions in C



ตัวอย่างการเขียนฟังก์ชั่นใน Arduino

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

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

Logical Operators

Logical operators allow you to combine logical expressions. The logical operators are presented in Table 6-1.

Table 6-1. Logical Operators

Type	Meaning	Example
&&	Logical AND	X && Y
||	Logical OR	X || Y
!	Logical NOT	!X

Logical AND Operator (&&)

Table 6-2. Logical AND (&&)

Expression1	Expression2	Expression1 && Expression2
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

เช่นถ้า k = 2 และ j = 3 แล้วเขียนตรวจสอบว่า

if (k == 2 && j == 3) เงื่อนไขนี้จะเป็นจริง

แต่ถ้าเขียนเปรียบเทียบว่า

if (k == 9 && j == 3) เงื่อนไขนี้จะเป็นเท็จ

Logical OR (||)

เช่นถ้า k = 2 และ j = 3 แล้วเขียนตรวจสอบว่า

if (k == 2 && j == 3) เงื่อนไขนี้จะเป็นจริง

หรือ

if (k == 9 || j == 3) เงื่อนไขนี้จะเป็นจริงเช่นกัน

แต่ถ้าเขียนเปรียบเทียบว่า

if (k == 9 && j == 5) เงื่อนไขนี้จะเป็นเท็จ

Logical NOT (!)

The logical NOT operator is the exclamation point (!). Because the logical NOT operator is a unary operator, its truth table is a little simpler, as shown in Table

Table 6-4. Logical NOT (!)

Expression1	! Expression1
TRUE	FALSE
FALSE	TRUE

ถ้ากำหนดให้

unsigned char k;

คำสั่งนี้จะเป็นจริงเมื่อไร ?

if (! (k == 2))

ถ้ากำหนดให้

unsigned char k;

และถ้าเขียนแบบนี้

if (! k == 2)

มีโอกาสเป็นจริงไหม ?