

Department of Computer Engineering
Faculty of Engineering
Prince of Songkla University

240-319

Embedded System Developer Module



March 23, 2023

Associate Prof. Dr. Panyayot Chaikan
panyayot@coe.psu.ac.th



Chapter 1

การเขียนโปรแกรมภาษาซีกับ AVR





เนื้อหา

แนะนำซอฟต์แวร์ Microchip Studio และ Arduino IDE

การดีบั๊กโปรแกรมภาษาซีโดยใช้ Microchip Studio

เปรียบเทียบภาษาแอสเซมบลีและภาษาซี

การเขียนโค้ดภาษาซี

ชนิดของข้อมูลพื้นฐานในภาษาซี

การเขียนโค้ดภาษาแอสเซมบลีแทรกในภาษาซี





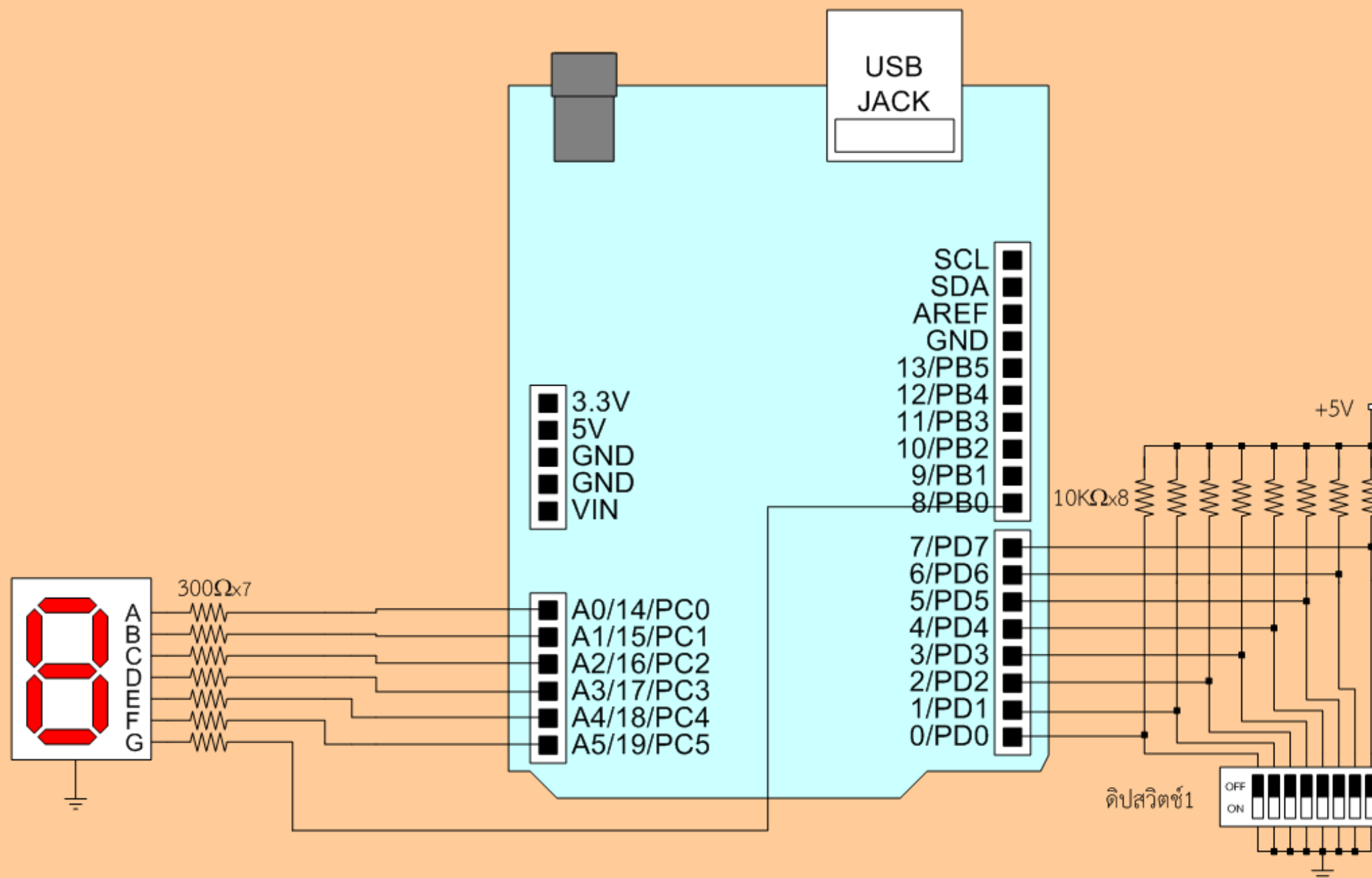
Three ways to program AVR microcontrollers

- ◆ Using assembly language
- ◆ Using C language with Microchip Studio's C compiler
- ◆ Using C language with the Arduino IDE





Example1 : counting logic high from DIP switch





How to solve Example1

- ◆ Next slides provide 3 ways to solve Example1
 - ◆ Utilizing assembly language
 - ◆ Utilizing C language with Microchip Studio
 - ◆ Utilizing C language with Arduino IDE



Solving Example1 using assembly

```
.include "m328Pdef.inc"
.equ    LED7SEG_PIN = 0x3F
.equ    ALL_PIN_IN = 0x00
.equ    PIN0_OUT = 0x01
.def    ZERO = R16
.def    VAR_A = R17
.def    COUNT = R18

.cseg
.org 0x00
    ldi    VAR_A, LED7SEG_PIN
    out    DDRC, VAR_A
    ldi    VAR_A, ALL_PIN_IN
    out    DDRD, VAR_A
    ldi    VAR_A, PIN0_OUT
    out    DDRB, VAR_A
    ldi    ZERO, 0x00
MAIN:
    in     VAR_A, PIND
    mov    COUNT, ZERO
```

```
LOOP:
    lsl    VAR_A
    adc    COUNT, ZERO
    cp     VAR_A, ZERO
    brne   LOOP
    call   DISPLAY_to_7SEGMENT
    rjmp   MAIN

DISPLAY_to_7SEGMENT:
    ldi    ZL, low(TB_7SEGMENT*2)
    ldi    ZH, high(TB_7SEGMENT*2)
    add    ZL, COUNT
    adc    ZH, ZERO
    lpm
    out    PORTC, r0
    rol    r0
    rol    r0
    rol    r0
    out    PORTB, r0
    ret
```





Solving Example1 using assembly

TB_7SEGMENT:

```
.DB      0b00111111, 0b00000110      ;0 และ 1      ----a----
.DB      0b01011011, 0b01001111      ;2 และ 3      f          b
.DB      0b01100110, 0b01101101      ;4 และ 5      ----g----
.DB      0b01111101, 0b00000111      ;6 และ 7      e          c
.DB      0b01111111, 0b01101111      ;8 และ 9      ----d----
```





Solving Example1 using C with Microchip Studio

```
#include <avr/io.h>
int main(void)
{
    unsigned char TABLE[] = {    0b00111111, 0b00000110,
                                   0b01011011, 0b01001111,
                                   0b01100110, 0b01101101,
                                   0b01111101, 0b00000111,
                                   0b01111111, 0b01101111
                                   };

    unsigned char count, tmp, i, test_bit;
    DDRD = 0x00;
    DDRB = 0x01;
    DDRC = 0x3F;
```





Solving Example1 using C with Microchip Studio

```
while (1)
{
    count = 0;
    tmp = PIND;
    for (i=0;i<8;i++)
    {
        test_bit = tmp & (0x01 << i);
        if (test_bit)
            count++;
    }
    tmp = TABLE[count];
    PORTC = tmp;
    PORTB = tmp >> 6;
}
}
```



Solving Example1 using C with Arduino IDE

```
const int segmentA = 14;
const int segmentB = 15;
const int segmentC = 16;
const int segmentD = 17;
const int segmentE = 18;
const int segmentF = 19;
const int segmentG = 8;
const int sw0 = 0;
const int sw7 = 7;

void display_0()
{
    digitalWrite(segmentA,HIGH);    digitalWrite(segmentB,HIGH);
    digitalWrite(segmentC,HIGH);    digitalWrite(segmentD,HIGH);
    digitalWrite(segmentE,HIGH);    digitalWrite(segmentF,HIGH);
    digitalWrite(segmentG,LOW);
}
```



Solving Example1 using C with Arduino IDE

```
void display_1()
{
    digitalWrite(segmentA, LOW);    digitalWrite(segmentB, HIGH);
    digitalWrite(segmentC, HIGH);   digitalWrite(segmentD, LOW);
    digitalWrite(segmentE, LOW);     digitalWrite(segmentF, LOW);
    digitalWrite(segmentG, LOW);
}

void display_2()
{
    digitalWrite(segmentA, HIGH);    digitalWrite(segmentB, HIGH);
    digitalWrite(segmentC, LOW);      digitalWrite(segmentD, HIGH);
    digitalWrite(segmentE, HIGH);     digitalWrite(segmentF, LOW);
    digitalWrite(segmentG, HIGH);
}
```



Solving Example1 using C with Arduino IDE

```
void display_3()
{
    digitalWrite(segmentA, HIGH);    digitalWrite(segmentB, HIGH);
    digitalWrite(segmentC, HIGH);    digitalWrite(segmentD, HIGH);
    digitalWrite(segmentE, LOW);     digitalWrite(segmentF, LOW);
    digitalWrite(segmentG, HIGH);
}

void display_4()
{
    digitalWrite(segmentA, HIGH);    digitalWrite(segmentB, HIGH);
    digitalWrite(segmentC, HIGH);    digitalWrite(segmentD, HIGH);
    digitalWrite(segmentE, HIGH);    digitalWrite(segmentF, HIGH);
    digitalWrite(segmentG, LOW);
}
```



Solving Example1 using C with Arduino IDE

```
void display_5()
{
    digitalWrite(segmentA, HIGH);    digitalWrite(segmentB, LOW );
    digitalWrite(segmentC, HIGH);    digitalWrite(segmentD, HIGH);
    digitalWrite(segmentE, LOW );    digitalWrite(segmentF, HIGH);
    digitalWrite(segmentG, HIGH);
}

void display_6()
{
    digitalWrite(segmentA, HIGH);    digitalWrite(segmentB, LOW );
    digitalWrite(segmentC, HIGH);    digitalWrite(segmentD, HIGH);
    digitalWrite(segmentE, HIGH);    digitalWrite(segmentF, HIGH);
    digitalWrite(segmentG, HIGH);
}
```



Solving Example1 using C with Arduino IDE

```
void display_7()
{
    digitalWrite(segmentA, HIGH );    digitalWrite(segmentB, HIGH );
    digitalWrite(segmentC, HIGH );    digitalWrite(segmentD, LOW  );
    digitalWrite(segmentE, LOW  );    digitalWrite(segmentF, LOW  );
    digitalWrite(segmentG, LOW  );
}

void display_8()
{
    digitalWrite(segmentA, HIGH );    digitalWrite(segmentB, HIGH );
    digitalWrite(segmentC, HIGH );    digitalWrite(segmentD, HIGH );
    digitalWrite(segmentE, HIGH );    digitalWrite(segmentF, HIGH );
    digitalWrite(segmentG, HIGH );
}
```



Solving Example1 using C with Arduino IDE

```
void display_9()
{
    digitalWrite(segmentA, HIGH);    digitalWrite(segmentB, HIGH);
    digitalWrite(segmentC, HIGH);    digitalWrite(segmentD, HIGH);
    digitalWrite(segmentE, LOW );    digitalWrite(segmentF, HIGH);
    digitalWrite(segmentG, HIGH);
}

int readSwitch()
{
    int count;
    count =0;
    for(int i=sw0; i<= sw7;i++)
    {
        if (digitalRead(i))
            count++;
    }
    return count;
}
```



Solving Example1 using C with Arduino IDE

```
void display7Segment(int value)
{
    switch(value)
    {
        case 0: display_0(); break;
        case 1: display_1(); break;
        case 2: display_2(); break;
        case 3: display_3(); break;
        case 4: display_4(); break;
        case 5: display_5(); break;
        case 6: display_6(); break;
        case 7: display_7(); break;
        case 8: display_8(); break;
        case 9: display_9(); break;
    }
}
```



Solving Example1 using C with Arduino IDE

```
void setup()
{
    for (int i = sw0; i <= sw7; i++)
    {
        pinMode(i, INPUT); //sw0-sw7 are connected to input pins
    }

    for (int i=segmentA; i<= segmentF; i++)
    {
        pinMode(i, OUTPUT); //segmentA-segmentF are set as output
    }
    pinMode(segmentG, OUTPUT);
}

//-----
void loop()
{
    int x;
    x = readSwitch();
    display7Segment(x);
}
```


Object code size of three different approaches

- ◆ Utilizing Assembly 62 bytes
- ◆ Utilizing C with Microchip Studio 276 bytes
- ◆ Utilizing C with Arduino IDE 1298 bytes
- ◆ Object code from Arduino IDE is 20.94 times larger than that from the assembly language
- ◆ Object code from Arduino IDE is 4.7 times larger than that from Microchip Studiio



Improved version of the C code on the Arduino IDE

```
const int zero[7] = {HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, LOW};
const int one[7] = {LOW, HIGH, HIGH, LOW, LOW, LOW, LOW};
const int two[7] = {HIGH, HIGH, LOW, HIGH, HIGH, LOW, HIGH};
const int three[7] = {HIGH, HIGH, HIGH, HIGH, LOW, LOW, HIGH};
const int four[7] = {LOW, HIGH, HIGH, LOW, LOW, HIGH, HIGH};
const int five[7] = {HIGH, LOW, HIGH, HIGH, LOW, HIGH, HIGH};
const int six[7] = {HIGH, LOW, HIGH, HIGH, HIGH, HIGH, HIGH};
const int seven[7] = {HIGH, HIGH, HIGH, LOW, LOW, LOW, LOW};
const int eight[7] = {HIGH, HIGH, HIGH, HIGH, HIGH, HIGH, HIGH};
const int nine[7] = {HIGH, HIGH, HIGH, HIGH, LOW, HIGH, HIGH};

const int* digits[10] = {zero, one, two, three, four, five, six, seven, eight, nine};

const int segmentA = 14;
const int segmentF = 19;
const int segmentG = 8;
const int sw0 = 0;
const int sw7 = 7;
```



Improved version of the C code on the Arduino IDE

```
int readSwitch()
{
    int count;
    count =0;
    for(int i=sw0; i<= sw7;i++)
    {
        if (digitalRead(i))
            count++;
    }
    return count;
}

void display7Segment(int value)
{
    for (int i = 0; i <= 5; i++)
    {
        digitalWrite(i + segmentA, digits[value][i]);
    }
    digitalWrite(segmentG, digits[value][6]);
}
```

Improved version of the C code on the Arduino IDE

```
//-----  
void setup()  
{  
  for (int i = sw0; i <= sw7; i++)  
  {  
    pinMode(i, INPUT); //sw0-sw7 are connected to input pins  
  }  
  
  for (int i=segmentA; i<= segmentF; i++)  
  {  
    pinMode(i, OUTPUT); //segmentA-segmentF are set as output  
  }  
  pinMode(segmentG, OUTPUT);  
}  
  
//-----  
void loop()  
{  
  int x;  
  x = readSwitch();  
  display7Segment(x);  
}
```



ขนาดของออบเจกต์โค้ดของโปรแกรมนที่ปรับใหม่

- ◆ ใช้ภาษา Assembly 62 bytes
- ◆ ใช้ภาษาซี บน Microchip Studio 276 bytes
- ◆ ใช้ภาษาซีบน Arduino IDE ตัวเดิม 1298 bytes
- ◆ ใช้ภาษาซีบน **Arduino IDE ตัวใหม่** **1268** bytes
- ◆ ใช้ภาษาซีบน Arduino IDE (ตัวใหม่) ได้ออบเจกต์โค้ดขนาดใหญ่กว่า Assembly 20.45 เท่า (ตัวเดิม 20.94 เท่า)
- ◆ ใช้ภาษาซีบน Arduino IDE ได้ออบเจกต์โค้ด (ตัวใหม่) ขนาดใหญ่กว่า ภาษาซีของ Microchip Studio 4.59 เท่า (ตัวเดิม 4.7 เท่า)



คุณสมบัติของโปรแกรมภาษาแอสเซมบลี และภาษาซี

คุณสมบัติ	ภาษาแอสเซมบลี	ภาษาซี
ความง่ายในการเรียนรู้	ยากสำหรับผู้เริ่มต้น	ง่ายกว่าแอสเซมบลี
ความเร็วในการพัฒนาโปรแกรม	เสียเวลามากกว่าใช้ภาษาซี	ทำได้เร็วกว่า ภาษาแอสเซมบลี
การนำโปรแกรมไปใช้งานใหม่ บนสถาปัตยกรรมอื่น	ทำไม่ได้โดยตรง	ทำได้โดยไม่ต้องดัดแปลง โปรแกรมมากนัก
ขนาดของโปรแกรมจุดหมาย (Object program)	เล็กกว่าภาษาซี	ใหญ่กว่าภาษาแอสเซมบลี
ความเร็วในการทำงานของโปรแกรม	เร็วกว่าภาษาซี	ช้ากว่าแอสเซมบลี



ข้อมูลของภาษาซีบน AVR และบนเครื่อง PC

ชนิดของข้อมูล	ภาษาซีบนเครื่องคอมพิวเตอร์ส่วนบุคคล		ภาษาซีบนสถาปัตยกรรมเอวีอาร์	
	จำนวน ไบต์	พิสัยของข้อมูลที่เก็บได้	จำนวน ไบต์	พิสัยของข้อมูลที่เก็บได้
char	1	-128 ถึง 127	1	-128 ถึง 127
unsigned char	1	0 ถึง 255	1	0 ถึง 255
int	4	-2,147,483,648 ถึง 2,147,483,647	2	-32,768 ถึง 32,767
unsigned int	4	0 ถึง 4,294,967,295	2	0 ถึง 65,535
short	2	-32,768 ถึง 32,767	2	-32,768 ถึง 32,767
unsigned short	2	0 ถึง 65,535	2	0 ถึง 65,535
long	4	-2,147,483,648 ถึง 2,147,483,647	4	-2,147,483,648 ถึง 2,147,483,647
unsigned long	4	0 ถึง 4,294,967,295	4	0 ถึง 4,294,967,295
float	4	$\pm 1.5 \times 10^{-45}$ ถึง $\pm 3.4 \times 10^{38}$	4	$\pm 1.5 \times 10^{-45}$ ถึง $\pm 3.4 \times 10^{38}$
double	8	$\pm 5.0 \times 10^{-324}$ ถึง $\pm 1.7 \times 10^{308}$	4	$\pm 1.5 \times 10^{-45}$ ถึง $\pm 3.4 \times 10^{38}$





Data types

```
typedef signed char ints_t;  
typedef unsigned char uints_t;  
typedef int int16_t;  
typedef unsigned int uint16_t;  
typedef long int32_t;  
typedef unsigned long uint32_t;  
typedef long long int64_t;  
typedef unsigned long long uint64_t;
```





ข้อมูลจำนวนเต็มในสถาปัตยกรรม AVR

ชนิดของข้อมูล	จำนวนไบต์	พิสัยของข้อมูลที่เก็บได้
uint8_t	1	0 ถึง 255
uint16_t	2	0 ถึง 65,535
uint32_t	4	0 ถึง 4,294,967,295
int8_t	1	-128 ถึง 127
int16_t	2	-32,768 ถึง 32,767
int32_t	4	-2,147,483,648 ถึง 2,147,483,647






คำสงวนในภาษาซี

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			



คำสั่งชี้แนะตัวประมวลผลก่อน



```
#define F_CPU          16000000
#define BAUD          38400
#define MYUBRR        F_CPU/16/BAUD-1
#define PI             3.14159265
#define my_name        "Panyayot Chaikan"
#define SWITCH PORTC

#undef BAUD
#undef MYUBRR
#undef PI
```





ตัวดำเนินการคำนวณ

สัญลักษณ์	ตัวดำเนินการ	ตัวอย่างการใช้งาน
+	การบวก	$C = A + B;$
-	การลบ	$C = A - B;$
*	การคูณ	$C = A * B;$
/	การหาร	$C = A / B;$
%	การมอดุลัส (Modulus) หรือการหารเอาเศษ	$C = A \% B;$
--	การลดค่า	$A--;$
++	การเพิ่มค่า	$B++;$

ตัวดำเนินการสัมพันธ์และตรรกะ



สัญลักษณ์	ตัวดำเนินการ	ตัวอย่างการใช้งานในข้อความสั่ง if เมื่อ a=4 และ b=5	
<	น้อยกว่า	if (a < b)	//เงื่อนไขที่ตรวจสอบเป็นจริง
<=	น้อยกว่าหรือเท่ากับ	if (a <= b)	//เงื่อนไขที่ตรวจสอบเป็นจริง
>	มากกว่า	if (a > b)	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
>=	มากกว่าหรือเท่ากับ	if (a >= b)	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
==	เท่ากับ	if (a == b)	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
!=	ไม่เท่ากับ	if (a != b)	//เงื่อนไขที่ตรวจสอบเป็นจริง
&&	และ	if ((a==b) && (a<0))	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
	หรือ	if ((a==b) (a>0))	//เงื่อนไขที่ตรวจสอบเป็นจริง





ตัวดำเนินการระดับบิต

สัญลักษณ์	ตัวดำเนินการ	ตัวอย่างการใช้งาน กรณีที่ $a = 0x70$ และ $b = 0x53$
\wedge	การดำเนินการเอกซก্লูซีฟ (Exclusive OR) ในระดับบิตต่อบิต	กำหนดให้ a, b, c มีขนาด 8 บิต $c = a \wedge b$; //ผลลัพธ์ $c = 0x23$
$\&$	การดำเนินการแอนด์ (AND) ในระดับบิตต่อบิต	$c = a \& b$; //ผลลัพธ์ $c = 0x50$
$ $	การดำเนินการออร์ (OR) ในระดับบิตต่อบิต	$c = a b$; //ผลลัพธ์ $c = 0x73$
\ll	เลื่อนบิตไปทางซ้าย	$c = a \ll 2$; //ผลลัพธ์ $c = 0xC0$
\gg	เลื่อนบิตไปทางขวา	$c = a \gg 2$; //ผลลัพธ์ $c = 0x1C$
\sim	กลับค่าบิตเป็นตรงข้าม	$c = \sim a$; //ผลลัพธ์ $c = 0x8F$





ตัวดำเนินการนิพจน์มีเงื่อนไข

นิพจน์ที่หนึ่ง ? นิพจน์ที่สอง : นิพจน์ที่สาม

เมื่อ

- ◆ นิพจน์ที่หนึ่ง คือ เงื่อนไขที่ต้องการตรวจสอบ
- ◆ นิพจน์ที่สอง คือ ข้อความสั่งที่จะถูกดำเนินการหากเงื่อนไข
นิพจน์ที่หนึ่งเป็นจริง
- ◆ นิพจน์ที่สาม คือ ข้อความสั่งที่จะถูกดำเนินการหากเงื่อนไขนิพจน์ที่หนึ่งเป็นเท็จ





ตัวดำเนินการนิพจน์มีเงื่อนไข : ตัวอย่าง

- ◆ $(a==b)? a++ : a--;$
- ◆ //ตรวจสอบว่า a เท่ากับ b หรือไม่ หากใช่ให้เพิ่มค่า a แต่หากไม่ใช่ให้ลดค่า a
- ◆ $(a<b)? (c=a+b) : (c=a-b);$
- ◆ //ตรวจสอบว่า a น้อยกว่า b หรือไม่ หากใช่ให้บวกค่าใน a และ b แล้วเก็บผลลัพธ์ใน c แต่หากไม่ใช่ให้ลบค่า b ออกจาก a แล้วเก็บผลลัพธ์ใน c



การเขียนโปรแกรมบนสถาปัตยกรรมAVR

- ◆ โค้ดภาษาใด ๆ ก็ตามจะต้องประกอบด้วย 2 ส่วน
- ◆ ส่วนตั้งค่าคอนฟิกของระบบ ซึ่งทำงานครั้งเดียว
- ◆ ส่วนวนลูปการทำงานไม่รู้จบ

การเขียนโปรแกรมแอสเซมบลีบนAVR

```
.org 0x00
```

```
    setup instruction 1
```

```
    setup instruction 2
```

```
    ...
```

```
    setup instruction n
```

```
MAIN:
```

```
    Instruction 1
```

```
    Instruction 2
```

```
    ...
```

```
    Instruction m
```

```
    rjmp MAIN
```



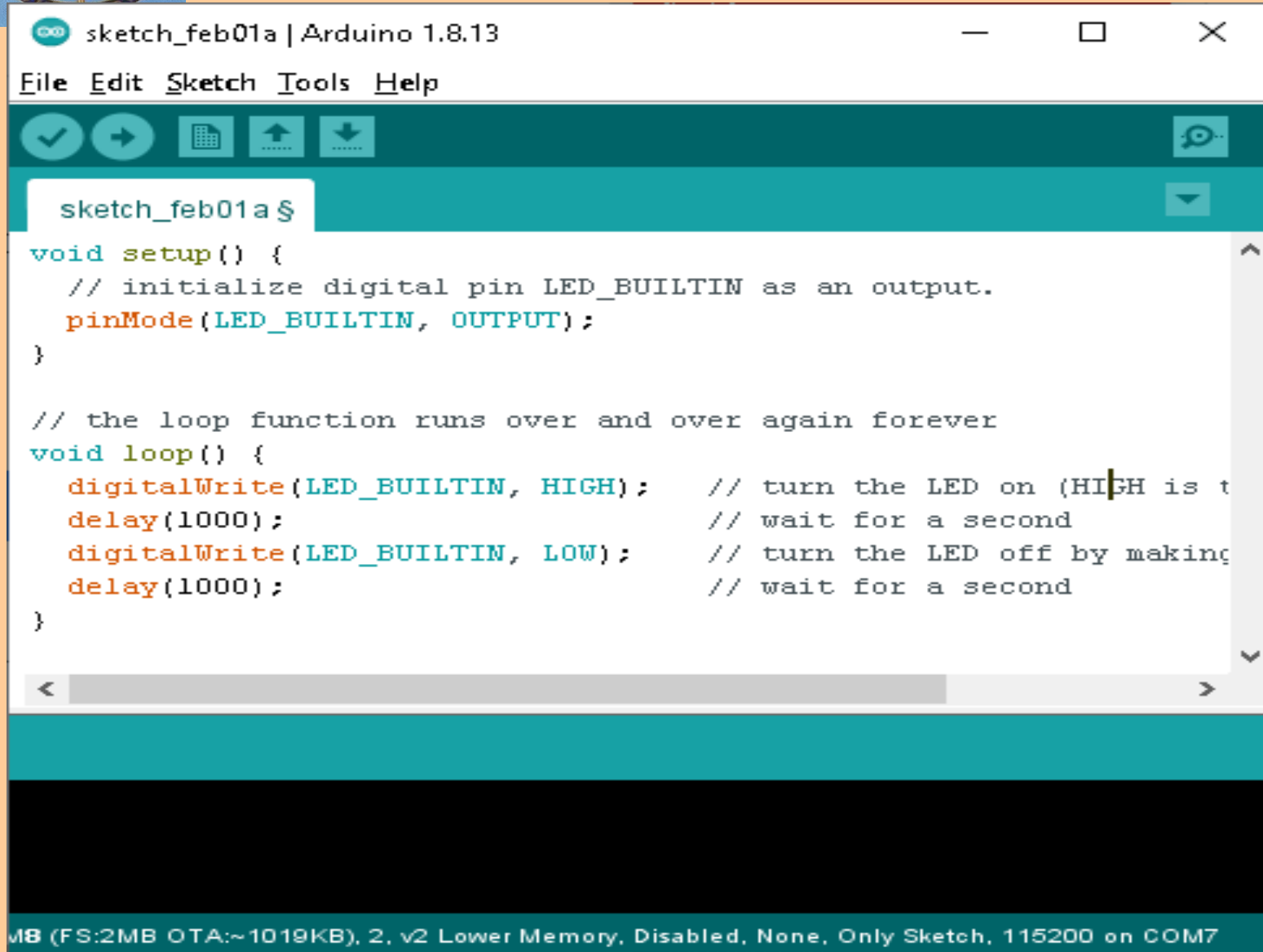


ภาษาซีของ AVR โดยใช้ Microchip Studio

```
#include <avr/io.h>
int main(void)
{
    คำสั่ง/ฟังก์ชันที่ต้องการกระทำการก่อนวนซ้ำ คำสั่ง/ฟังก์ชันที่ 1
    คำสั่ง/ฟังก์ชันที่ต้องการกระทำการก่อนวนซ้ำ คำสั่ง/ฟังก์ชันที่ 2
    .....
    คำสั่ง/ฟังก์ชันที่ต้องการกระทำการก่อนวนซ้ำ คำสั่ง/ฟังก์ชันที่ n
    while(1)
    {
        ... ;           //คำสั่งหรือฟังก์ชันที่ต้องการเรียกในวงวน คำสั่ง/ฟังก์ชันที่ 1
        ... ;           //คำสั่งหรือฟังก์ชันที่ต้องการเรียกในวงวน คำสั่ง/ฟังก์ชันที่ 2
        ...
        ... ;           //คำสั่งหรือฟังก์ชันที่ต้องการเรียกในวงวน คำสั่ง/ฟังก์ชันที่ n
    }
}
```



ภาษา C บนแพลตฟอร์ม Arduino IDE



```
sketch_feb01a | Arduino 1.8.13
File Edit Sketch Tools Help

sketch_feb01a §

void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the positive voltage)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the pin LOW
  delay(1000);                     // wait for a second
}

1MB (FS:2MB OTA:~1019KB), 2, v2 Lower Memory, Disabled, None, Only Sketch, 115200 on COM7
```

```
#include <WProgram.h>
int main(void)
{
    init();
    setup();
    for (;;)
        loop();
    return 0;
}
```

- ◆ ฟังก์ชัน main ถูก Arduino IDE ซ่อนไว้ ผู้ใช้เพียงเขียน 2 ฟังก์ชัน คือ setup และ loop



การวนซ้ำไม่รู้จบในภาษาซี

```
while(1)
{
    ....    //คำสั่งในวงวนคำสั่งที่ 1
    ....    //คำสั่งในวงวนคำสั่งที่ 2
    ....
    ....    //คำสั่งในวงวนคำสั่งที่ n
}
```

```
for (;;)
{
    ....    //คำสั่งในวงวนคำสั่งที่ 1
    ....    //คำสั่งในวงวนคำสั่งที่ 2
    ....
    ....    //คำสั่งในวงวนคำสั่งที่ n
}
```



การวนซ้ำไม่รู้จบในภาษาซี (ต่อ)

```
do
{
    ....    //คำสั่งในวงวนคำสั่งที่ 1
    ....    //คำสั่งในวงวนคำสั่งที่ 2
    ....
    ....    //คำสั่งในวงวนคำสั่งที่ n
}
while (1);
```





แนะนำภาษา C

Basic C Program



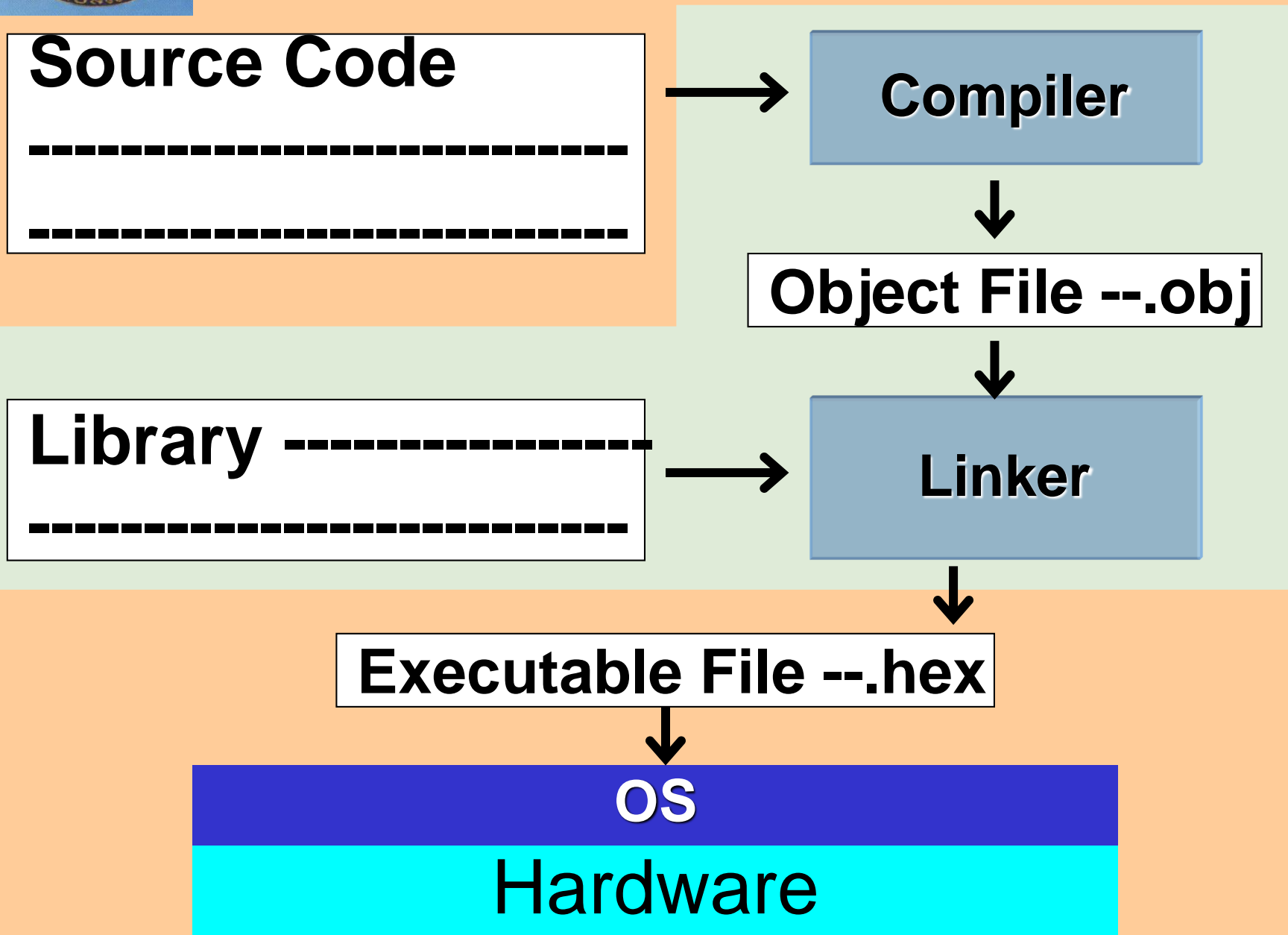


ภาษาไทย *C Programming Language*

- ◆ ถูกออกแบบและสร้างขึ้นในปี พ.ศ. 2515 (1972) โดย Dennis Ritchie
 - ◆ เป็นภาษาเชิงโครงสร้าง (structured programming) โดยมีการนิยามและเรียกใช้ฟังก์ชัน
 - ◆ มีโครงสร้างและลำดับการเขียนที่มีความยืดหยุ่น
 - ◆ สามารถเข้าถึงอุปกรณ์ฮาร์ดแวร์ได้โดยตรง
 - ◆ สามารถขยายโปรแกรมเป็น C++



ขั้นตอนการพัฒนาโปรแกรม



- ◆ มีการแปล Source Code ทั้งหมดก่อนได้ Object File
- ◆ นำมาประกอบกับ Code บางส่วนที่เตรียมไว้แล้ว (Library)
- ◆ ได้ผลลัพธ์เป็น machine language (Executable File)

โครงสร้างโปรแกรมในภาษา C

```
# include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    // My first program
```

```
    printf("Hello World!!!");
```

```
    system("PAUSE");
```

```
    return 0;
```

```
}
```

Pre-processor

ฟังก์ชัน main

Comment คำอธิบายโปรแกรม

คำสั่ง หรือ ฟังก์ชัน

ขอบเขตของฟังก์ชัน main



โครงสร้างโปรแกรมในภาษา C ใน Arduino IDE

```
# include <stdio.h>
int main (void)
{
    setup();
    while (1)
    {
        loop();
    }
    return 0;
}
```





Arduino Programming Convention

```
void setup()          //ฟังก์ชัน setup
{
    //ข้อความสั่ง1
    //ข้อความสั่ง2
    ...
    //ใส่ข้อความสั่งที่ต้องการให้ตัวประมวลผลทำงานเพียงหนึ่งครั้งเมื่อเริ่มทำงานไว้ในฟังก์ชันนี้
}

void loop()           //ฟังก์ชัน loop
{
    //ข้อความสั่ง1
    //ข้อความสั่ง2
    ...
    //ใส่ข้อความสั่งที่ต้องการให้ตัวประมวลผลทำงานซ้ำแบบไม่รู้จบไว้ในฟังก์ชันนี้
}
```



ส่วนของ Preprocessor

- ◆ บรรทัดที่เริ่มต้นด้วย เครื่องหมาย # เรียกว่า preprocessor ตัวอย่างเช่น
 - ◆ `#define symbol another_symbol`
 - ◆ ใช้ในการแทนที่คำที่กำหนด (symbol) ทุกคำในซอร์สโค้ดด้วยคำที่กำหนด (another_symbol)
 - ◆ มักใช้ในการกำหนดค่าคงที่ เพื่อสะดวกต่อการแก้ไข
 - ◆ เช่น ในไฟล์ซอร์สโค้ดมีโค้ดที่เกี่ยวกับกับเลข 50 ซึ่งใช้ในการตัดเกรด ซึ่งมีการใช้ในหลายจุดของโปรแกรม เราใช้ #define เพื่อให้ง่ายต่อการเปลี่ยนแปลง
 - ◆ `#define FAIL_SCORE 50`
 - ◆ `#define VAT 7`





ส่วนของ Preprocessor

- ◆ `#include <stdio.h>` ซึ่งเป็นตัวบอกว่าให้นำ header file ของมาตรฐานการส่งรับข้อมูล ใน `stdio.h` เข้ามาในโปรแกรมด้วย เนื่องจากจำเป็นต้องใช้
- ◆ โดย header file ตัวนี้มีข้อมูลของฟังก์ชัน `printf()` ที่ใช้ในโปรแกรมของเรา



int main (void)

- ◆ ทุกๆโปรแกรมจะต้องมีฟังก์ชันที่ชื่อว่า main. เพราะนี่เป็นส่วนแรกที่โปรแกรมจะเริ่มทำงาน
- ◆ ฟังก์ชัน main() ถูกกำหนดขึ้นให้เป็นส่วนแรกของโปรแกรมที่จะถูกรันเพื่อให้สะดวกแก่การอ่านโปรแกรม ทุกโปรแกรมต้องมีฟังก์ชันที่มีชื่อนี้ และต้องมีเพียงฟังก์ชันเดียวเท่านั้นเพื่อให้โปรแกรมสามารถ compile และทำงานได้





int main (void)

- ◆ คำสงวน “int” ที่นำหน้า main() เป็นตัวบอกว่าฟังก์ชันจะมีการส่งค่าออกเป็นชนิดตัวเลขจำนวนเต็ม (int มาจาก integer) เมื่อจบฟังก์ชัน
- ◆ วงเล็บที่ตามหลังคำสงวน “main” บอกให้ทราบว่านี่คือฟังก์ชัน
- ◆ คำสงวน “void” บอกให้ทราบว่าฟังก์ชันดังกล่าวไม่มีการรับค่าเข้าจากภายนอก (parameter/argument)
- ◆ int main() มีความหมายเดียวกับ int main(void)





ส่วนของโปรแกรม

- ◆ วงเล็บปีกกาเปิด { บอกให้ทราบว่าส่วนของฟังก์ชันหรือขอบเขตของโค้ดเริ่มต้นที่ใด
- ◆ วงเล็บปีกกาปิด } บอกให้ทราบว่าส่วนของฟังก์ชันหรือขอบเขตของโค้ดสิ้นสุดที่ใด
- ◆ เครื่องหมาย ; บอกให้ทราบว่าคำสั่งหนึ่งจบลง (ต้องใส่ไว้หลังคำสั่งทุกคำสั่ง มิฉะนั้นจะเกิด syntax error)
- ◆ การย่อหน้าในการเขียนโปรแกรมเป็นลักษณะการเขียนโปรแกรมที่ดี เพื่อให้สะดวกต่อการหาขอบเขตของคำสั่ง





return 0 ;

- ◆ เนื่องจากฟังก์ชัน main() มีการส่งค่ากลับมา ก่อนจะจบฟังก์ชันจึงต้องมีการใช้คำสั่ง return ตามด้วยตัวเลขที่จะส่งกลับไป โดยค่าที่ส่งกลับไปจะถูกส่งไปให้ระบบปฏิบัติการ (operating systems)
- ◆ ค่า 0 เป็นตัวบอกว่าโปรแกรมทำงานเสร็จสิ้นสมบูรณ์
- ◆ ยังไม่ต้องกังวลกับเรื่องนี้ตอนนี้ และในสภาพแวดล้อม Arduino IDE ก็ไม่ได้มีการใช้ฟังก์ชัน main โดยตรง





ตัวแปรและตัวดำเนินการ ในภาษาซี



ชนิดข้อมูลพื้นฐาน

- ◆ ภาษา C เป็นภาษาที่เข้มงวดกับเรื่องชนิดข้อมูล
 - ◆ int ตัวเลขจำนวนเต็ม
 - ◆ float ตัวเลขทศนิยม
 - ◆ double ตัวเลขทศนิยมที่มีความจุเป็น 2 เท่า
 - ◆ char ตัวอักษร
- ◆ ชนิดข้อมูลบางตัว อาจทำงานร่วมกับตัวปรับปรุง signed, unsigned, short และ long





ค่าคงที่ Constant Value

- ◆ ตัวเลขจำนวนเต็ม (ฐานสิบ Decimal) เช่น 10, 20, -5
 - ◆ เลขฐาน 16 (Hexadecimal) เช่น 0x32, 0x5FB (ขึ้นต้นด้วย 0x)
 - ◆ เลขฐาน 8 (Octal) เช่น 013, 041, 07 (ขึ้นต้นด้วยศูนย์)
- ◆ ตัวเลขทศนิยม เช่น 7.2, 5.6, 0.002, 2e-3, -3.14159e2, -314.159
- ◆ ตัวอักษร เช่น 'c' , '1' , '5' , ' ' (space), '\$'
 - ◆ ตัวอักษรพิเศษที่ขึ้นต้นด้วย \ เช่น '\t'
 - ◆ อักขระพิเศษสามารถยกเลิกความพิเศษได้โดยใช้ \ นำหน้า เช่น '\\' คืออักขระ \
- ◆ ข้อความ String (ไม่ใช่ข้อมูลชนิดพื้นฐาน) เช่น "Hello"
 - ◆ ข้อความว่างเปล่า ""
 - ◆ ข้อความที่มีตัวอักษรว่าง(space)หนึ่งตัว " "





ข้อมูลตัวอักษร Character

- ขนาดหนึ่งไบต์ เก็บค่าเป็นจำนวนเต็มได้ 256 ค่า (0_255 หรือ -128_127)
- ค่าจำนวนคือรหัสของตัวอักษรตามมาตรฐานASCII เรียกว่ารหัสแอสกี (ASCII)

Dec	Char/Description	Dec	Char	Dec	Char	Dec	Char
0	null	33	!	65	A	97	a
1	start of heading	34	"	66	B	98	b
2	start of text	35	#	67	C	99	c
3	end of text	36	\$	68	D	100	d
4	end of trans. block	37	%	69	E	101	e
5	enquiry	38	&	70	F	102	f
6	acknowledge	39	'	71	G	103	g
7	bell	40	(72	H	104	h
8	backspace	41)	73	I	105	i
9	horizontal tab	42	^	74	J	106	j
10	new line	43	+	75	K	107	k
11	vertical tab	44	,	76	L	108	l
12	new page	45	-	77	M	109	m
13	carriage return	46	.	78	N	110	n
14	shift out	47	/	79	O	111	o
15	shift in	48	0	80	P	112	p
16	data link escape	49	1	81	Q	113	q
17	device control 1	50	2	82	R	114	r
18	device control 2	51	3	83	S	115	s
19	device control 3	52	4	84	T	116	t
20	device control 4	53	5	85	U	117	u
21	neg acknowledge	54	6	86	V	118	v
22	synchronous idle	55	7	87	W	119	w
23	end of trans. block	56	8	88	X	120	x
24	cancel	57	9	89	Y	121	y
25	end of medium	58	:	90	Z	122	z
26	substitute	59	;	91	[123	{
27	escape	60	<	92	\	124	
28	file separator	61	=	93]	125	}
29	group separator	62	>	94	^	126	~
30	record separator	63	?	95	_	127	DEL
31	unit separator	64	@	96	`		





ตัวแปร Variable

- ◆ ใช้หน่วยความจำของคอมพิวเตอร์ในการจดจำข้อมูล
- ◆ ยากที่จะอ้างอิงตำแหน่งในหน่วยความจำ เราจึงใช้ชื่อของตัวแปรในการอ้างอิง
- ◆ ตัวแปรต้องถูกประกาศก่อนการใช้งาน
- ◆ การประกาศตัวแปร ประกอบด้วย ชนิดข้อมูล ชื่อตัวแปร (อาจจะมีการกำหนดค่าเริ่มต้นด้วย) และตามด้วย ; (semi-colon)
 - เช่น `int x;` หรือ `int x = 2;`
 - ชื่อตัวแปร เป็นการประกอบกันระหว่าง ตัวอักษร ตัวเลข เครื่องหมาย `_` (underscore) ทั้งนี้ห้ามขึ้นต้นด้วยตัวเลข และไม่เป็นคำสงวน





ตัวอย่างการประกาศใช้งานตัวแปร

- ◆ ประกาศตัวแปร c มีชนิดเป็นตัวอักษร 1 ตัว

char c;

- ◆ ประกาศตัวแปร count มีชนิดเป็นเลขจำนวนเต็มพร้อมทั้งกำหนดให้มีค่าเท่ากับ 8

int count=8;

- ◆ ประกาศตัวแปร price มีชนิดเป็นเลขทศนิยม

float price;





ตัวอย่างการประกาศตัวแปร

- ประกาศตัวแปรชื่อ happy มีชนิดเป็นตัวอักษร 1 ตัว พร้อมทั้งกำหนดให้มีค่าเป็นตัวอักษร C (capital letter พิมพ์ใหญ่)

```
char happy='C';
```

- ประกาศตัวแปร easy มีชนิดเป็นเลขจำนวนเต็มพร้อมทั้งกำหนดให้มีค่าเท่ากับ 5 และตัวแปร Easy มีชนิดเป็นเลขจำนวนเต็มพร้อมทั้งกำหนดให้มีค่าเท่ากับ 10

```
int easy=5, Easy=10;      หรือ
```

```
int easy=5; int Easy=10;
```

ประกาศตัวแปรชื่อ test มีชนิดเป็นเลขทศนิยมมีค่าเท่ากับ 7.5

```
float test=7.5;
```



การตั้งชื่อตัวแปร

◆ อักษรตัวแรกจะต้องเป็นตัวอักษรภาษาอังกฤษหรือเครื่องหมาย

_ (underline character)

■ สามารถตั้งชื่อตัวแปรโดยใช้ตัวเลขร่วมกับตัวอักษรภาษาอังกฤษ ได้

แต่ห้ามใช้ตัวเลขเป็นตัวอักษรตัวแรก

■ ตัวอักษรภาษาอังกฤษตัวพิมพ์เล็กและตัวพิมพ์ใหญ่ถือเป็นคนละตัวกัน

เช่น name กับ NAME





การตั้งชื่อตัวแปร

◆ ห้ามตั้งชื่อซ้ำกับคำสงวน (Keywords) ซึ่งเป็นคำที่มีอยู่แล้วใน
ภาษา C

auto	default	float	register	struct
volatile	break	do	for	return
switch	while	case	double	goto
short	typedef	char	else	if
signed	union	const	enum	int
sizeof	unsigned	continue	extern	long
static	void			





ขอบเขตของตัวแปร

- ◆ ตัวแปรถูกประกาศได้ทั้งภายนอก และภายในฟังก์ชัน
- ◆ ตัวแปรที่ถูกประกาศภายในฟังก์ชัน
 - ◆ ถูกสร้างขึ้น เมื่อฟังก์ชันเริ่มทำงาน
 - ◆ ถูกทำลาย เมื่อฟังก์ชันจบการทำงาน
 - ◆ เป็นที่รู้จัก ภายในเครื่องหมาย { และ } ของฟังก์ชันที่มีการประกาศเท่านั้น
 - ◆ ไม่สามารถอ้างอิงจากฟังก์ชันอื่นได้ แม้ว่า ตัวแปรนั้นยังไม่ได้ถูกทำลาย
- ◆ ตัวแปรที่ถูกประกาศภายนอกฟังก์ชัน
 - ◆ ถูกสร้างขึ้น และดำรงอยู่ ตลอดการทำงานของโปรแกรม
 - ◆ เป็นที่รู้จัก ในทุกๆ ฟังก์ชันภายในไฟล์เดียวกัน
 - ◆ สามารถรู้จักกันต่างไฟล์ได้ โดยใช้ตัวปรับปรุง extern



ตัวอย่างการใช้ตัวแปรภายใน

```
// internal.c
#include <stdio.h>
void my_func();
char
int main()
{
    double x = 1.1;
    my_func();
    printf("In main x = %f\n",x);
    return 0;
}
void my_func()
{
    double x;
    x = 2.5;
    printf("In my_func x = %f\n",x);
}
```


ตัวอย่างการใช้ตัวแปรภายนอก

```
//external.c
#include <iostream.h>
double x;
void my_func();
int main()
{
    x = 1.1;
    my_func();
    printf("In main x = %f\n",x);
    return 0;
}
void my_func()
{
    x = 2.5;
    printf("In my_func x = %f\n",x);
}
```





ตัวแปรภายในที่มีชื่อเดียวกับตัวแปรภายนอก

```
// variable.c
#include <stdio.h>
double x;
void my_func();
int main()
{
    double x = 1.1;
    my_func();
    printf("In main x = %f\n",x);
    return 0;
}
void my_func()
{
    x = 2.5;
    printf("In my_func x = %f\n",x);
}
```



ข้อมูลของภาษาซีบนแต่ละสถาปัตยกรรม

ชนิดของข้อมูล	ภาษาซีบนเครื่องคอมพิวเตอร์ส่วนบุคคล		ภาษาซีบนสถาปัตยกรรมเอวีอาร์	
	จำนวน ไบต์	พิสัยของข้อมูลที่เก็บได้	จำนวน ไบต์	พิสัยของข้อมูลที่เก็บได้
char	1	-128 ถึง 127	1	-128 ถึง 127
unsigned char	1	0 ถึง 255	1	0 ถึง 255
int	4	-2,147,483,648 ถึง 2,147,483,647	2	-32,768 ถึง 32,767
unsigned int	4	0 ถึง 4,294,967,295	2	0 ถึง 65,535
short	2	-32,768 ถึง 32,767	2	-32,768 ถึง 32,767
unsigned short	2	0 ถึง 65,535	2	0 ถึง 65,535
long	4	-2,147,483,648 ถึง 2,147,483,647	4	-2,147,483,648 ถึง 2,147,483,647
unsigned long	4	0 ถึง 4,294,967,295	4	0 ถึง 4,294,967,295
float	4	$\pm 1.5 \times 10^{-45}$ ถึง $\pm 3.4 \times 10^{38}$	4	$\pm 1.5 \times 10^{-45}$ ถึง $\pm 3.4 \times 10^{38}$
double	8	$\pm 5.0 \times 10^{-324}$ ถึง $\pm 1.7 \times 10^{308}$	4	$\pm 1.5 \times 10^{-45}$ ถึง $\pm 3.4 \times 10^{38}$



คำสงวนในภาษาซี

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			





คำสั่งชี้แนะตัวประมวลผลก่อน (Preprocessor)

```
#define F_CPU 16000000
#define BAUD 38400
#define MYUBRR F_CPU/16/BAUD-1
#define PI 3.14159265
#define my_name "Panyayot Chaikan"
#define SWITCH PORTC

#undef BAUD
#undef MYUBRR
#undef PI
```





ตัวดำเนินการคำนวณ

สัญลักษณ์	ตัวดำเนินการ	ตัวอย่างการใช้งาน
+	การบวก	$C = A + B;$
-	การลบ	$C = A - B;$
*	การคูณ	$C = A * B;$
/	การหาร	$C = A / B;$
%	การมอดุลัส (Modulus) หรือการหารเอาเศษ	$C = A \% B;$
--	การลดค่า	$A--;$
++	การเพิ่มค่า	$B++;$

ตัวดำเนินการสัมพันธ์และตรรกะ



สัญลักษณ์	ตัวดำเนินการ	ตัวอย่างการใช้งานในข้อความสั่ง if เมื่อ a=4 และ b=5	
<	น้อยกว่า	if (a < b)	//เงื่อนไขที่ตรวจสอบเป็นจริง
<=	น้อยกว่าหรือเท่ากับ	if (a <= b)	//เงื่อนไขที่ตรวจสอบเป็นจริง
>	มากกว่า	if (a > b)	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
>=	มากกว่าหรือเท่ากับ	if (a >= b)	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
==	เท่ากับ	if (a == b)	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
!=	ไม่เท่ากับ	if (a != b)	//เงื่อนไขที่ตรวจสอบเป็นจริง
&&	และ	if ((a==b) && (a<0))	//เงื่อนไขที่ตรวจสอบเป็นเท็จ
	หรือ	if ((a==b) (a>0))	//เงื่อนไขที่ตรวจสอบเป็นจริง

ตัวดำเนินการระดับบิต

สัญลักษณ์	ตัวดำเนินการ	ตัวอย่างการใช้งาน กรณีที่ $a = 0x70$ และ $b = 0x53$
\wedge	การดำเนินการเอกซกลูซีฟ (Exclusive OR) ในระดับบิตต่อบิต	กำหนดให้ a, b, c มีขนาด 8 บิต $c = a \wedge b$; //ผลลัพธ์ $c = 0x23$
$\&$	การดำเนินการแอนด์ (AND) ในระดับบิตต่อบิต	$c = a \& b$; //ผลลัพธ์ $c = 0x50$
$ $	การดำเนินการออร์ (OR) ในระดับบิตต่อบิต	$c = a b$; //ผลลัพธ์ $c = 0x73$
$<<$	เลื่อนบิตไปทางซ้าย	$c = a << 2$; //ผลลัพธ์ $c = 0xC0$
$>>$	เลื่อนบิตไปทางขวา	$c = a >> 2$; //ผลลัพธ์ $c = 0x1C$
\sim	กลับค่าบิตเป็นตรงข้าม	$c = \sim a$; //ผลลัพธ์ $c = 0x8F$



ตัวดำเนินการนิพจน์มีเงื่อนไข

นิพจน์ที่หนึ่ง ? นิพจน์ที่สอง : นิพจน์ที่สาม

เมื่อ

- ◆ นิพจน์ที่หนึ่ง คือ เงื่อนไขที่ต้องการตรวจสอบ
- ◆ นิพจน์ที่สอง คือ ข้อความสั่งที่จะถูกดำเนินการหากเงื่อนไข
นิพจน์ที่หนึ่งเป็นจริง
- ◆ นิพจน์ที่สาม คือ ข้อความสั่งที่จะถูกดำเนินการหากเงื่อนไขนิพจน์ที่หนึ่งเป็นเท็จ



ตัวดำเนินการนิพจน์มีเงื่อนไข : ตัวอย่าง

- ◆ $(a == b) ? a++ : a--;$
- ◆ //ตรวจสอบว่า a เท่ากับ b หรือไม่ หากใช่ให้เพิ่มค่า a แต่หากไม่ใช่ให้ลดค่า a
- ◆ $(a < b) ? (c = a + b) : (c = a - b);$
- ◆ //ตรวจสอบว่า a น้อยกว่า b หรือไม่ หากใช่ให้บวกค่าใน a และ b แล้วเก็บผลลัพธ์ใน c แต่หากไม่ใช่ให้ลบค่า b ออกจาก a แล้วเก็บผลลัพธ์ใน c





คำสั่ง if-else

if (เงื่อนไข)
คำสั่ง

ตัวอย่าง

```
if (x==y)  
x++;
```

```
if (เงื่อนไข)  
คำสั่ง  
else  
คำสั่ง
```

ตัวอย่าง

```
if (x==y)  
x++;  
else  
y++;
```





คำสั่ง if-else

```
if (x==y)
```

```
{
```

```
    x++;
```

```
    y=y+100;
```

```
}
```

```
if (x==y)
```

```
{
```

```
    x++;
```

```
    y=y+100;
```

```
}
```

```
else
```

```
{
```

```
    y++;
```

```
    x+= 5;
```

```
}
```

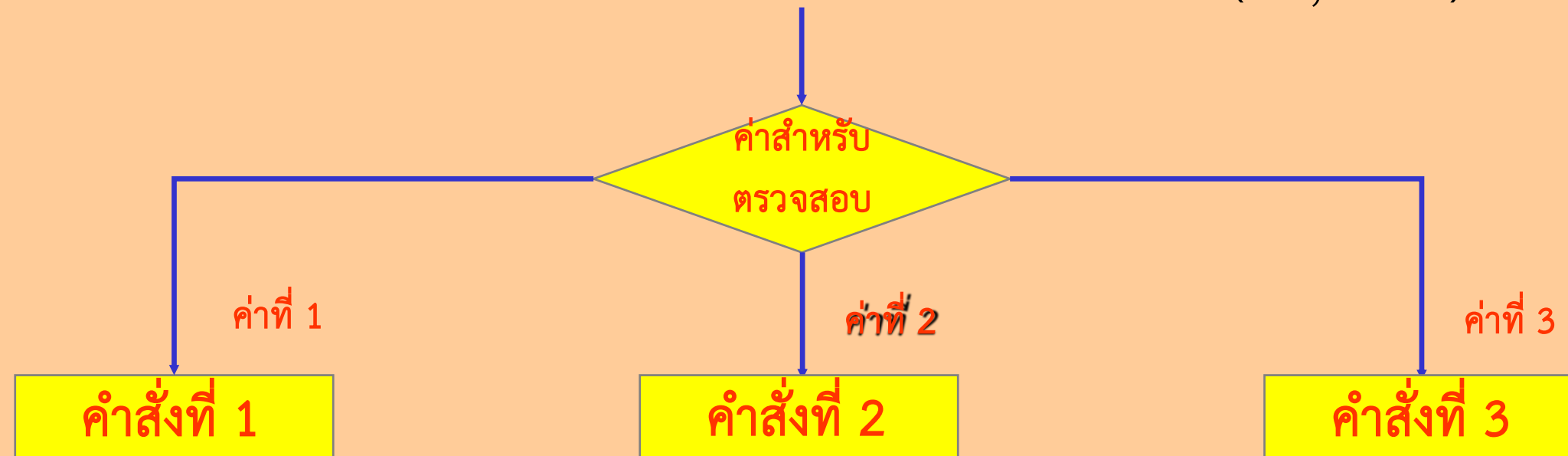




คำสั่ง switch-case

ใช้ควบคุมให้โปรแกรมเลือกดำเนินการไปในเส้นทางใดเส้นทางหนึ่ง (case) จากทางเลือกหลาย ๆ ทาง โดยใช้ค่าที่ต้องการตรวจสอบว่าตรงกับค่าใด

- ◆ นิพจน์ที่อยู่ในคำสั่ง switch จะถูกตรวจสอบว่าตรงกับ case ไດ แล้วโปรแกรมก็จะทำงานตามคำสั่งที่อยู่ใน case นั้น และคำสั่งอื่น ๆ ที่ตามมาจนจบโครงสร้าง switch-case หรือเจอคำสั่ง break ก็จะออกจากโครงสร้าง switch-case
- ◆ โดยค่าที่ใช้ตรวจสอบ จะต้องเป็นจำนวนเต็มหรือตัวอักษรเท่านั้น (int, char)





โครงสร้างแบบทางเลือก switch-case

```
switch ( นิพจน์ที่ต้องการตรวจสอบ )
```

```
{
```

```
    case ค่าที่ 1 :      คำสั่งที่ 1;
```

```
    case ค่าที่ 2 :      คำสั่งที่ 2;
```

```
    case ค่าที่ 3 :      คำสั่งที่ 3;
```

```
    case      ...  ...
```

```
}
```

switch และ *case* คือคำสั่งวน

นิพจน์ที่อยู่ในคำสั่ง switch จะถูกตรวจสอบตามลำดับว่าตรงกับ case ไດ โปรแกรมก็จะทำงานตามคำสั่งที่อยู่ใน case นั้น รวมถึงคำสั่งอื่น ๆ ที่ตามมา ใน case ที่เหลือจนจบโครงสร้าง switch-case หรือเจอคำสั่ง break ก็จะออกจากโครงสร้าง switch-case ได้



```

#include<stdio.h>
setup()
{
    int c;
    printf("Enter integer 1 or 2 or 3:");
    scanf("%d",&c);
    switch(c)
    {
        case 1: printf("ONE\n");

        case 2: printf("TWO\n");

        case 3: printf("THREE\n");

    }
}

```

จะใช้คำสั่ง break; มาช่วย

ผลลัพธ์

Enter integer 1 or 2 or 3: 3
THREE

ผลลัพธ์

Enter integer 1 or 2 or 3: 2
TWO
THREE ← ไม่ต้องการ

ผลลัพธ์

Enter integer 1 or 2 or 3: 1
ONE
TWO
THREE } ไม่ต้องการ



คำสั่ง *break*

- ◆ คำสั่ง `break` จะใช้สำหรับการควบคุมการกระทำ โดยบังคับการกระทำ
- ◆ บ่อยครั้งที่จะใช้คำสั่ง `break` เป็นคำสั่งสุดท้ายในแต่ละ `case`
- ◆ หากไม่มีคำสั่ง `break` ในชุดคำสั่งของ `case` ใด โปรแกรมจะทำงานต่อไป ในคำสั่งของทุกๆ `case` ถัดไปด้วยจนจบ

`switch` (นิพจน์ที่ต้องการตรวจสอบ)

```
{  
    case   ค่าที่ 1 :   คำสั่งที่ 1; break;  
  
    case   ค่าที่ 2 :   คำสั่งที่ 2; break;  
  
    case   ค่าที่ 3 :   คำสั่งที่ 3; break;  
  
    case   ... .. break;  
  
}
```





```
#include<stdio.h>
int main()
{
    int c;
    printf("Enter integer 1 or 2 or 3:");
    scanf("%d",&c);
    switch(c)
    {
        case 1: printf("ONE\n");
        break;
        case 2: printf("TWO\n");
        break;
        case 3: printf("THREE\n");
        break;
    }
}
```

ผลลัพธ์

Enter integer 1 or 2 or 3:3
THREE

ผลลัพธ์

Enter integer 1 or 2 or 3:2
TWO

ผลลัพธ์

Enter integer 1 or 2 or 3:1
ONE



คำสั่งเงื่อนไข switch-case



- ◆ default เป็นค่าสงวน
- ◆ default เป็นอีกกรณีหนึ่งในคำสั่ง switch-case มักวางไว้เป็นกรณีสุดท้าย
- ◆ ในกรณีที่ตรวจสอบแล้วพบว่า นิพจน์ มีค่าไม่ตรงกับ case ใด ๆ เลย ข้างต้น โปรแกรมจะเข้าไปทำงานในส่วนของ default
- ◆ ไม่จำเป็นต้องใส่ break หลังชุดคำสั่งของ default






```
#include<stdio.h>
int main()
{
    int c;
    printf("Enter integer 1 or 2 or 3 ");
    scanf("%d",&c);
    switch(c)
    {
        case 1: printf("ONE\n");
                break;
        case 2: printf("TWO\n");
                break;
        case 3: printf("THREE\n");
                break;
        default: printf("Out of range");
    }
}
```

ผลลัพธ์

Enter integer 1 or 2 or 3 **4**
Out of range





```

#include<stdio.h>
int main()
{
    char grade;
    printf("Enter your grade: ");
    scanf("%c",&grade);
    switch(grade)
    {
        case 'a':
        case 'A': printf("Very Good\n");
                break;

        case 'b':
        case 'B': printf("Good\n");
                break;

        case 'c':
        case 'C': printf("Fair\n");
                break;

        default: printf("No good!\n");
    }
}

```

ผลลัพธ์

Enter your grade: a
Very good

Enter your grade: F
No good!

Enter your grade: C
Fair

Enter your grade: 3
No good!



if-else vs switch-case

◆ **switch-case** ใช้ในกรณีที่มีทางเลือกหลายทาง โดยขึ้นอยู่กับค่าของตัวแปร(หรือนิพจน์)หนึ่ง ที่มีค่าเป็น int หรือ char เท่านั้น

◆ **if-else** ใช้ตรวจสอบเงื่อนไข ได้หลากหลายกว่า ตามต้องการ (เช่น เปรียบเทียบค่าน้อยกว่า มากกว่า หรืออยู่ในช่วงใดช่วงหนึ่ง รวมถึงการใช้ logical operator สร้างเงื่อนไขได้ซับซ้อนขึ้น)แต่ถ้ามีหลายทางเลือก ต้องใช้ if ซ้อนกันหลายๆชั้น หรือสร้างเงื่อนไขที่ซับซ้อนขึ้น

◆ บางกรณีสามารถแปลงคำสั่ง switch-case เป็น if-else ได้ เช่น

```
switch(a)
{ case b: คำสั่ง_1; break;
  case c: คำสั่ง_2; break;
  case d: คำสั่ง_3; break;
  default: คำสั่ง_4;
}
```

```
if (a==b)
    คำสั่ง_1;
else if (a==c)
    คำสั่ง_2;
else if (a==d)
    คำสั่ง_3;
else คำสั่ง_4;
```



จบ if-else และ switch-case

จงเขียนโปรแกรม รับตัวอักษรหนึ่งตัว แล้วตรวจสอบว่าเป็น สระ(vowel) หรือ พยัญชนะ (consonant) สมมติว่าผู้ใช้ใส่เฉพาะตัวอักษร a-z เท่านั้น

◆ ให้เติมส่วนของโปรแกรมนี้ให้สมบูรณ์

```
char ch;
```

```
printf("Enter a character(a-z): ");
```

```
scanf("%c", &ch);
```

เติมเต็มโปรแกรมในส่วนนี้ จะเลือกใช้โครงสร้าง if-else หรือ switch-case ก็ได้

คำแนะนำ ให้ตรวจสอบค่า ch ว่าเป็นตัวอักษร สระ a e i o u หรือไม่ ถ้าใช่ก็พิมพ์ vowel ถ้าไม่ใช่ ก็ให้พิมพ์ว่าเป็น Consonant

```
printf("End of the program.\n");
```





คำสั่ง if-else

```
if (x==y)
```

```
    x++;
```

```
    y=y+100;
```

ควรหลีกเลี่ยง

```
if (x==y)
```

```
    x++;
```

```
    y=y+100;
```

เขียนแบบนี้ดีกว่า

การวนลูป



◆ for

◆ while

◆ do-while





คำสั่งการทำซ้ำ **for**

for (การกำหนดค่าเริ่มต้นตัวแปร ; เงื่อนไข ; ปรับค่าตัวแปร)

คำสั่ง; หรือ กลุ่มคำสั่ง

จากตัวอย่างก่อนหน้านี้ สามารถเขียนโปรแกรมให้อยู่ในรูปของ for loop ได้ดังนี้
พิมพ์ Hello สองร้อยครั้ง

```
int i;  
for (i=0; i<200; i=i+1)    printf("Hello");
```

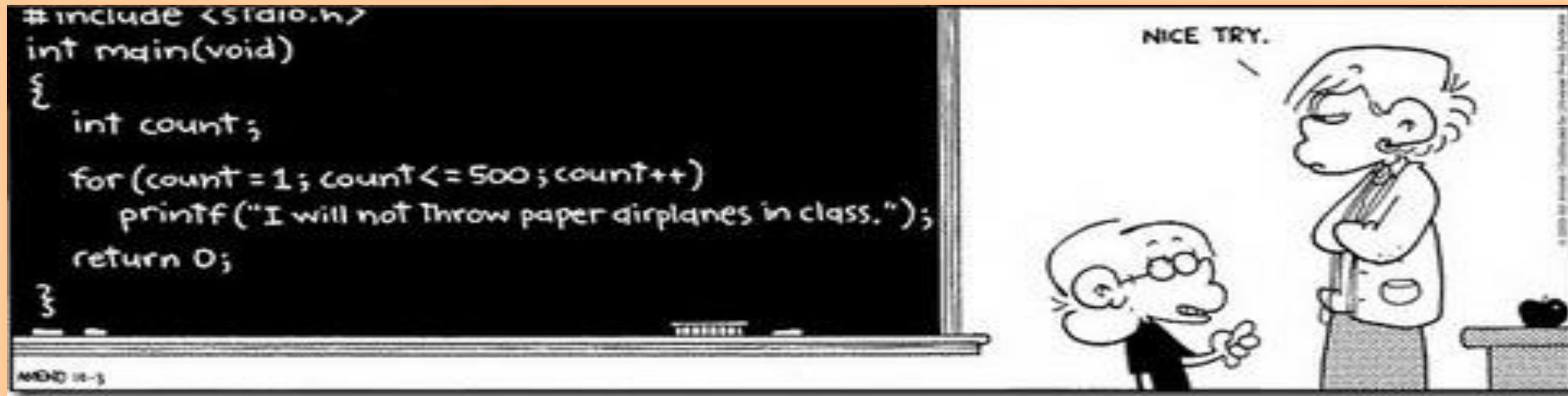
พิมพ์ตารางการคูณ แม่สอง

```
int i;  
for (i=1; i<=12; i++ )  
{    printf("2 x %d = %d \n",i, 2*i); }
```





การทำซ้ำ (ซ้ำชั้น)



เด็ก: เสร็จแล้วค่ะครู

ครู : ... พยายามดีเหลือเกิน (กะจะลองดีกับฉันใช่ไหม?!)

เด็กฉลาด (แกมโกง)

ครูทำโทษให้เด็กคัดบนกระดาน "ฉันจะไม่เล่นปาเครื่องบินกระดาษในห้องเรียนอีกแล้ว" 500 จบ แต่เด็ก(หัวใส) เขียนเป็นโค้ดโปรแกรมภาษาซีที่จะทำให้ได้ผลลัพธ์เหมือนกัน





คำสั่งการทำซ้ำ **for**

- หากเงื่อนไขเป็นจริง loop ก็将继续ทำงานต่อไป จนกระทั่งเงื่อนไขเป็นเท็จ ตัวอย่างเช่น
- การปรับค่าตัวแปร (ลด/เพิ่ม) อาจจะทำให้เกิดทำงานไม่รู้จบของ loop ได้ (infinite loop)

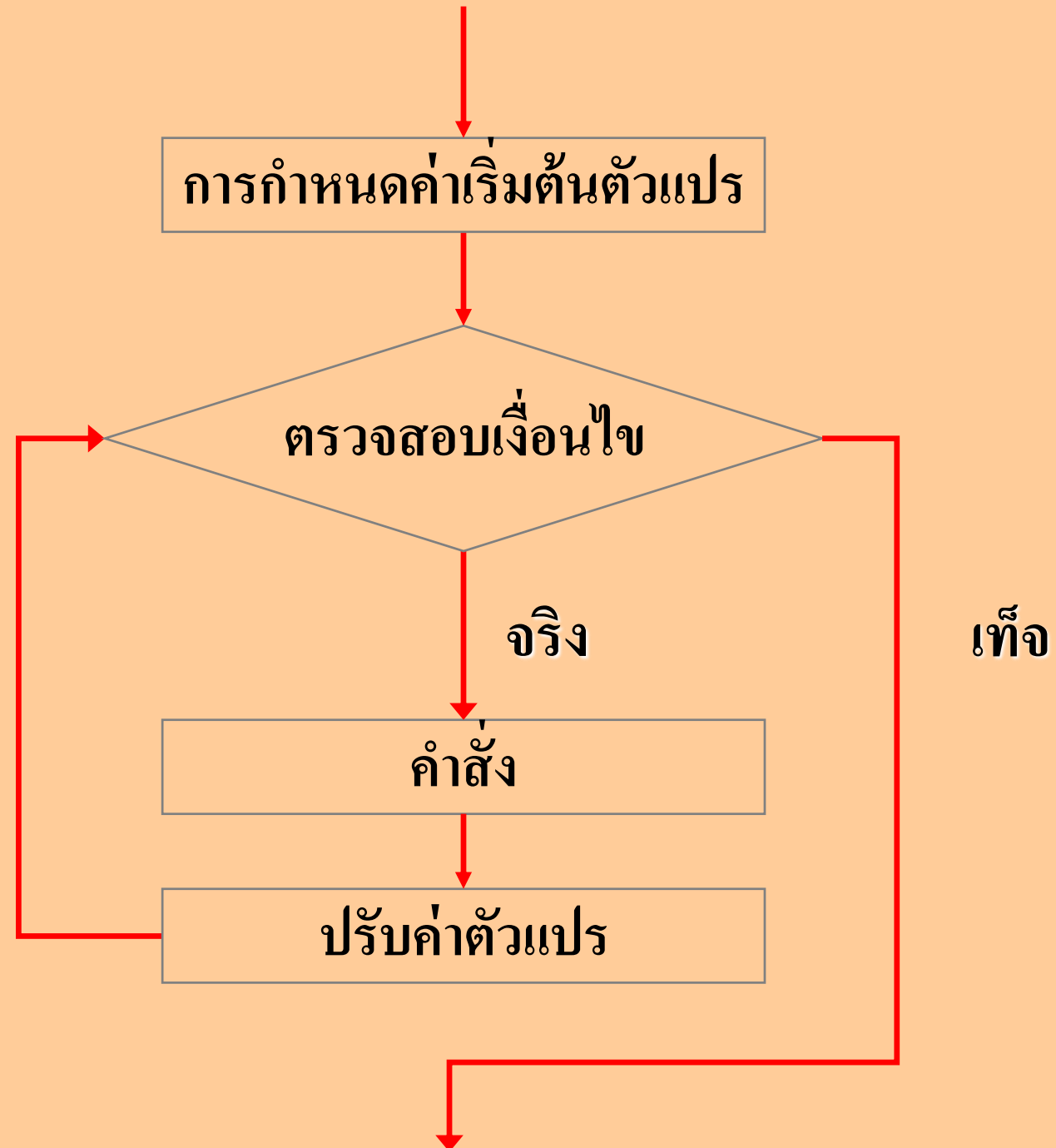
```
#include<stdio.h>
int main()
{
    int i;
    for (i=0;i<5;i--)
        printf("* \n");
}
```

เกิดการวนลูปไม่รู้จบ





Flow chart การทำงาน คำสั่งการทำซ้ำ for





การใช้งานคำสั่ง for

- มักใช้ ในกรณี รู้จำนวนรอบการทำงานซ้ำ ที่แน่นอน หรือ มีตัวนับจำนวนรอบ (counter) กำกับ
- ตัว counter ต้องถูกกำหนดค่าเริ่มต้น, และมีการปรับเปลี่ยนค่า
- ควรระวังการกำหนดเงื่อนไขการทำงานซ้ำ ที่จะต้องกลายเป็นเท็จได้ เพื่อไม่ทำให้เกิดลูปอนันต์ (infinite loop)
- การกำหนด จำนวนรอบ อย่างง่าย

ถ้าต้องการทำ n รอบ

```
int i,n;  
for (i=0; i<n; i++)  
{ printf("%d \n",i); }
```

```
int i,n;  
for (i=1; i<=n; i++)  
{ printf("%d \n",i); }
```




คำสั่งการทำซ้ำ while

รูปแบบของ while loop คือ

while (เงื่อนไข)

while เป็นคำสั่งวน

คำสั่ง;

หากเงื่อนไขเป็น**จริง** โปรแกรมจะทำงานตามคำสั่งซ้ำ จนกระทั่งเงื่อนไขเป็น**เท็จ**

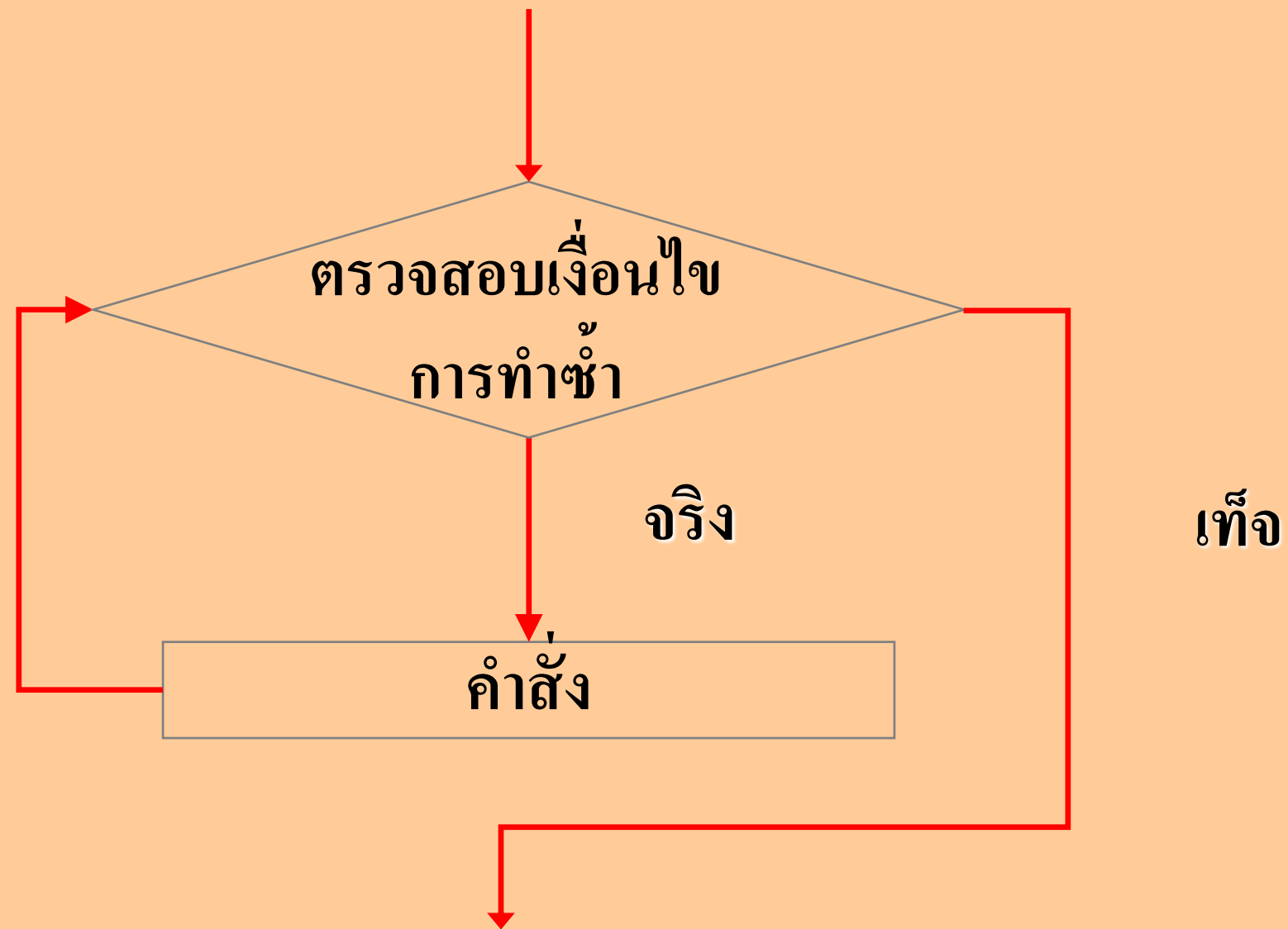
ในขณะที่ **เงื่อนไข** เป็นจริง
ให้ทำคำสั่ง

หรือ ทำคำสั่งนั้นจนกว่า
เงื่อนไขจะกลายเป็นเท็จ





Flowchart คำสั่ง while



คำสั่งการทำซ้ำ while

for loop

```
#include<stdio.h>
int main()
{
    int i;
    for (i=0; i<5; i++)
        printf("* \n");
}
```

while loop

```
#include<stdio.h>
int main()
{
    int i=0;
    while (i<5)
    {
        printf("* \n");
        i++;
    }
}
```





เปรียบเทียบโครงสร้างของ for loop กับ while loop

for loop

for (การกำหนดค่าเริ่มต้นตัวแปร ; **เงื่อนไข** ; ปรับค่าตัวแปร)

คำสั่ง;

while loop

การกำหนดค่าเริ่มต้นตัวแปร;

while (**เงื่อนไข**){

คำสั่ง;

ปรับค่าตัวแปร;

}





คำสั่งการทำซ้ำ **do-while**

- ◆ โครงสร้างแบบการทำซ้ำ while และ for จะต้องมีการตรวจสอบค่าของเงื่อนไขก่อนว่าเป็นจริงหรือเท็จ ก่อนที่จะทำคำสั่ง (กลุ่มคำสั่ง) ภายในรอบ
- ◆ ถ้าต้องทำคำสั่งภายในรอบก่อนอย่างน้อย 1 ครั้ง แล้วจึงตรวจสอบเงื่อนไข เมื่อจบรอบ ให้ใช้โครงสร้าง **do-while**





โครงสร้าง do while loop

do

{

กลุ่มคำสั่ง ;

}

while (เงื่อนไข) ;

มี ; เพื่อจบคำสั่ง
do-while





ตัวอย่าง do while loop

```
#include<stdio.h>
int main()
{
    int i=6;
    do
    {
        printf("* \n");
        i++;
    }
    while (i<5);
}
```

ลูปนี้ทำกี่รอบ?

รอบเดียว

ถ้าต้องการให้ทำ 6 รอบ ต้องแก้ไข อย่างไร

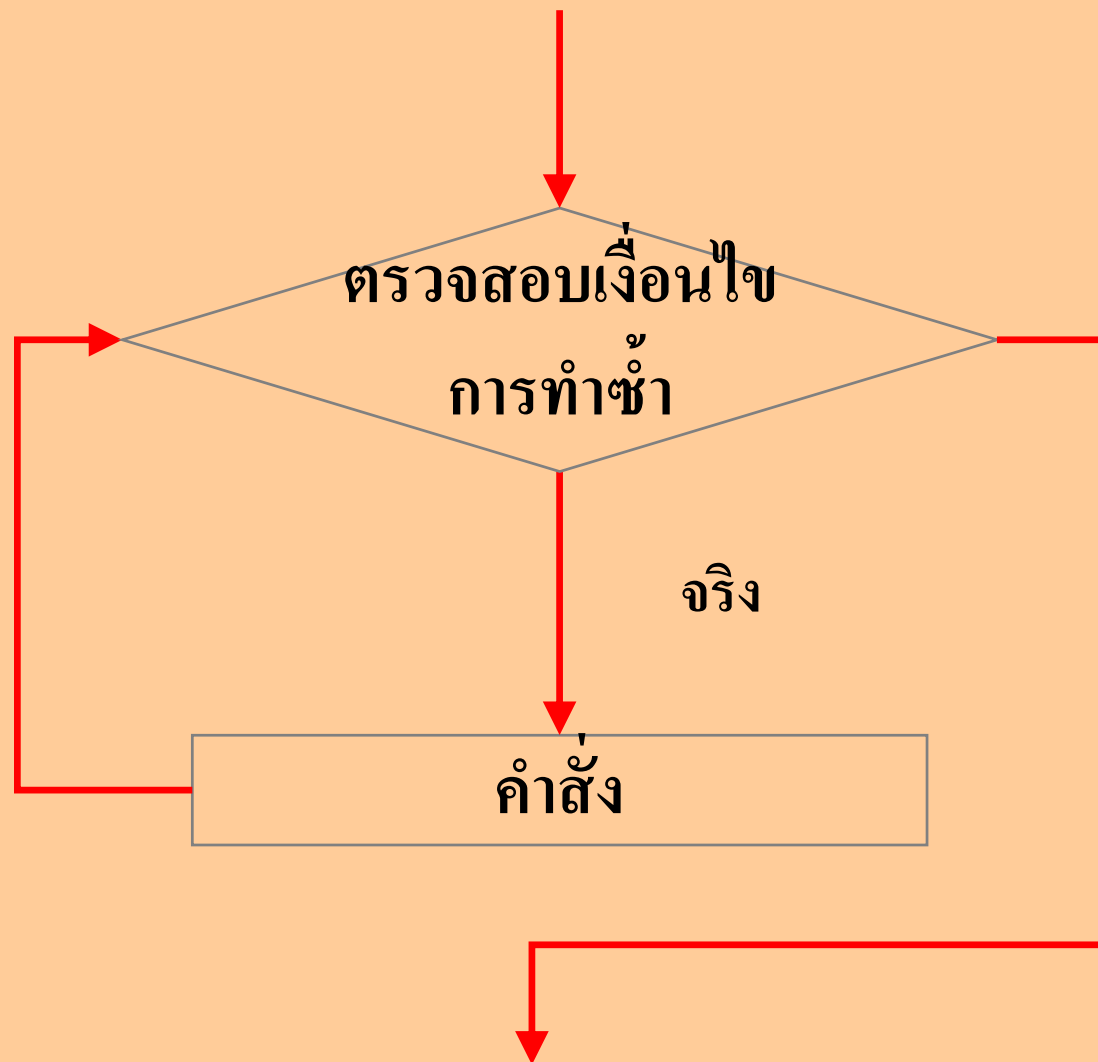
```
#include<stdio.h>
int main()
{
    int i=6;
    do
    {
        printf("* \n");
        i++;
    }
    while (i<12);
}
```



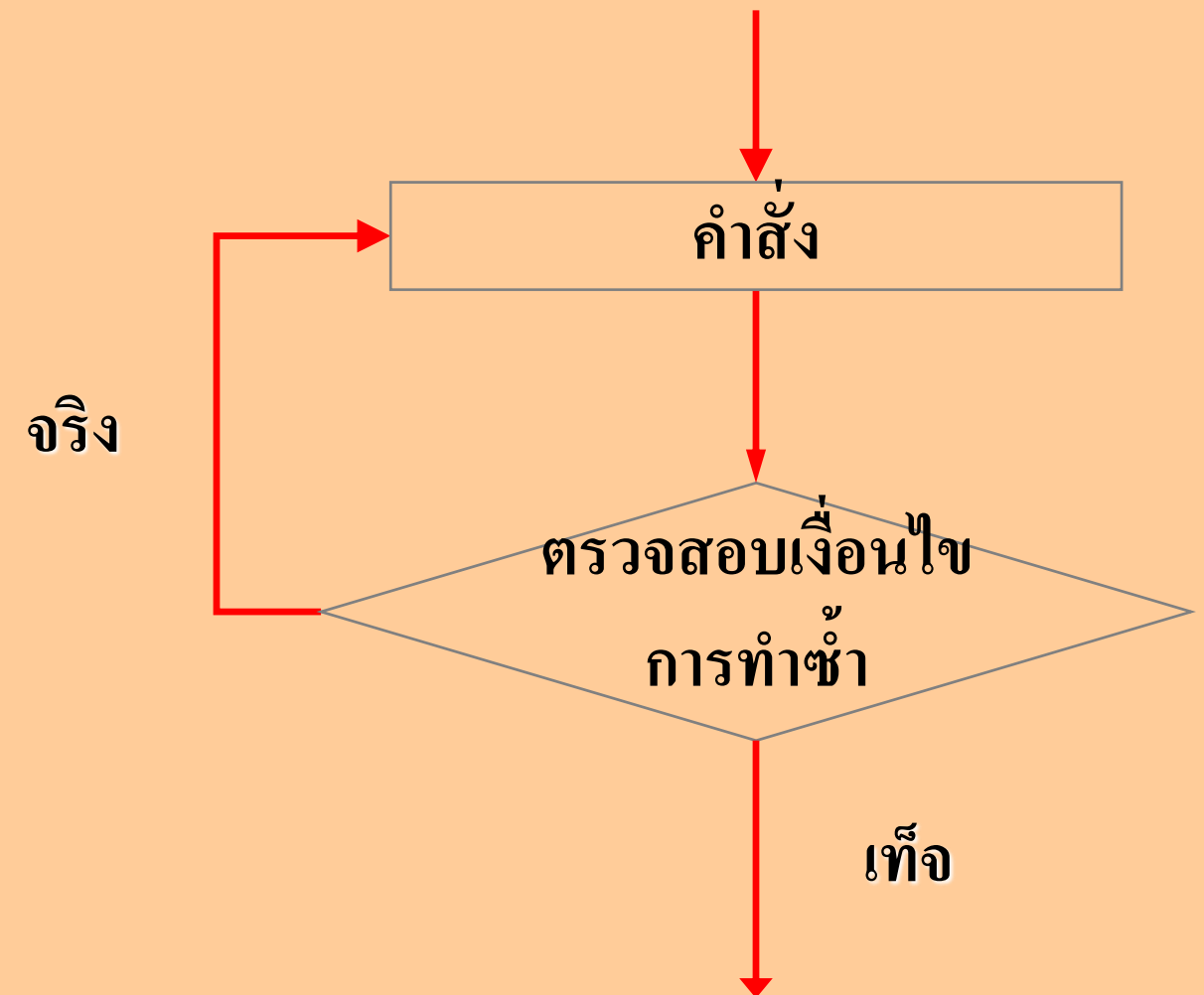


เปรียบเทียบโครงสร้าง do while กับ while loop

while loop



do loop





คลังโปรแกรม (Library) ที่ใช้งานบ่อย บน Microchip Studio

ชื่อคลังโปรแกรม	สิ่งที่บรรจุไว้ในคลังโปรแกรม
<alloca.h>	ฟังก์ชันสำหรับการจัดสรรพื้นที่ในสแต็ก
<ctype.h>	ฟังก์ชันสำหรับตรวจสอบชนิด ขนาด และรูปแบบของตัวอักษร
<math.h>	ค่าคงที่และฟังก์ชันทางคณิตศาสตร์ที่จำเป็นต้องใช้บ่อย
<string.h>	ใช้สำหรับจัดการข้อมูลชนิดสายอักขระ (String)
<avr/boot.h>	โปรแกรมอรรถประโยชน์สำหรับการบูตโหลดเดอร์ (Boot loader)
<avr/eeprom.h>	ฟังก์ชันสำหรับจัดการการอ่านเขียนหน่วยความจำอีอีพร็อม
<avr/fuse.h>	ค่าควบคุมสำหรับเข้าถึงฟิวส์ควบคุมหน้าที่การทำงานของตัวประมวลผล

คลังโปรแกรม (Library) ที่ใช้งานบ่อย บน Microchip Studio



ชื่อคลังโปรแกรม	สิ่งที่บรรจุไว้ในคลังโปรแกรม
<avr/io.h>	รายชื่อเรจิสเตอร์และบิตของข้อมูลพื้นฐานของสถาปัตยกรรมเอวีอาร์
<avr/interrupt.h>	ใช้สำหรับสนับสนุนกระบวนการขัดจังหวะตัวประมวลผล
<util/delay.h>	ฟังก์ชันสำหรับหน่วงเวลา
<avr/lock.h>	สำหรับเข้าถึงบิตที่ใช้ปิดกั้นการคัดลอกโปรแกรมในหน่วยความจำแฟลช
<avr/power.h>	ฟังก์ชันและค่าควบคุมสำหรับการจัดการการใช้พลังงานในตัวประมวลผล
<avr/sfr_defs.h>	บรรจุรายชื่อเรจิสเตอร์ใช้งานเฉพาะด้าน
<avr/sleep.h>	ฟังก์ชันและค่าควบคุมสำหรับจัดการการเข้าสู่ภาวะหลับของตัวประมวลผล
<avr/wdt.h>	ฟังก์ชันและค่าควบคุมตัวจับเวลารีเซ็ตดีออก



การเรียกใช้เรจิสเตอร์ไอโอในภาษาซี

PINB	TIFR2	TCCR0A	SMCR	PCMSK1
DDRB	PCIFR	TCCR0B	MCUSR	PCMSK2
PORTB	EIFR	TCNT0	MCUCR	TIMSK0
PINC	EIMSK	OCR0A	SPMCSR	TIMSK1
DDRC	GPIOR0	OCR0B	WDTCR	TIMSK2
PORTC	EECR	GPIOR1	CLKPR	ADC
PIND	EEDR	GPIOR2	PRR	ADCL
DDRD	EEAR	SPCR	OSCCAL	ADCH
PORTD	EEARL	SPSR	PCICR	ADCSRA
TIFR0	EEARH	SPDR	EICRA	ADCSRB
TIFR1	UDR0	ACSR	PCMSK0	ADMUX



การเรียกใช้เรจิสเตอร์ไอโอในภาษาซี (ต่อ)

DIDR0	OCR1AH	TWSR
DIDR1	OCR1B	TWAR
TCCR1A	OCR1BL	TWDR
TCCR1B	OCR1BH	TWCR
TCCR1C	TCCR2A	TWAMR
TCNT1	TCCR2B	UCSR0A
TCNT1L	TCNT2	UCSR0B
TCNT1H	OCR2A	UCSR0C
ICR1	OCR2B	UBRR0
OCR1A	ASSR	UBRR0L
OCR1AL	TWBR	UBRR0H



การกำหนดทิศทางของพอร์ต

```
#include <avr/io.h>
```

```
char a = 0xFF;           //กำหนดตัวแปร a ขนาด 8 บิตหนึ่งตัว
```

```
//ให้มีค่าเริ่มต้นเท่ากับ 0xFF
```

```
DDRA = 0x00;           //กำหนดให้พอร์ต A ทำหน้าที่เป็นพอร์ตรับเข้า
```

```
DDRB = 0xFF;           //กำหนดให้พอร์ต B ทำหน้าที่เป็นพอร์ตส่งออก
```

```
DDRC = a;              //กำหนดให้พอร์ต C ทำหน้าที่เป็นพอร์ตส่งออก
```

```
DDRD = 0x00;           //กำหนดให้พอร์ต D ทำหน้าที่เป็นพอร์ตรับเข้า
```





การอ่านข้อมูลจากพอร์ตอินพุต

```
char b, c;           //กำหนดตัวแปรขนาด 8 บิต จำนวน 2 ตัว  
                     //ได้แก่ b และ c  
  
b = PINA;           //อ่านค่าสถานะทางตรรกะจากพอร์ต A  
                     //นำค่าที่ได้ใส่ในตัวแปร b  
  
c = PIND;           //อ่านค่าสถานะทางตรรกะจากพอร์ต D  
                     //นำค่าที่ได้ใส่ในตัวแปร c
```



การเขียนข้อมูลสู่พอร์ตเอาต์พุต

```
char d;           //กำหนดตัวแปรขนาด 8 บิตหนึ่งตัว  
                  //คือ ตัวแปร d  
  
d = 0x20;         //ให้ตัวแปร d มีค่าเท่ากับ 0x20;  
  
PORTB = d;        //ส่งค่าจากตัวแปร d ให้กับพอร์ต B  
  
PORTC = 0x50;     //ส่งค่า 0x50 ให้กับพอร์ต C โดยตรง
```





การเข้าถึงพอร์ตแบบ Read-Modify-Write

PORTC += 1;

//เพิ่มค่าในพอร์ต C ขึ้นหนึ่งค่า

PORTC = PORTC +1;

//เพิ่มค่าในพอร์ต C ขึ้นหนึ่งค่า (อีกวิธีหนึ่ง)

PORTC++;

//เพิ่มค่าในพอร์ต C ขึ้นหนึ่งค่า (อีกวิธีหนึ่ง)



ซอฟต์แวร์ที่ใช้

- ◆ AVR Studio (เวอร์ชันเก่าจะใช้ชื่อนี้)
- ◆ Atmel Studio
- ◆ Microchip Studio (เวอร์ชันล่าสุดใช้ชื่อนี้)



การดีบั๊กโปรแกรมด้วย AVRStudio

Processor

Name	Value
Program Counter	0x00007E
Stack Pointer	0x085D
X pointer	0x0075
Y pointer	0x085F
Z pointer	0x026C
Cycle Counter	211
Frequency	4.0000 MHz
Stop Watch	52.75 us
SREG	1 1 1 1 1 1 1 1
Registers	

```
void ONE_MINUTE_PASSED(void)

//-----
void main(void)
{
    SET_PORT_DIRECTION();
    SET_TIMER1_MODE();
    RESET_MINUTE_COUNTER();
    cli();

    unsigned char PREVIOUS_SW_SAMPLING, CURRENT_SW_SAMPLING;

    CURRENT_STATE = RELAY_OFF;
}
```

Watch

Name	Value	Type	Location
M_CNT1	0x00 ''	unsigned char	0x0072 [SRAM]
M_CNT0	0x00 ''	unsigned char	0x0074 [SRAM]

I/O View

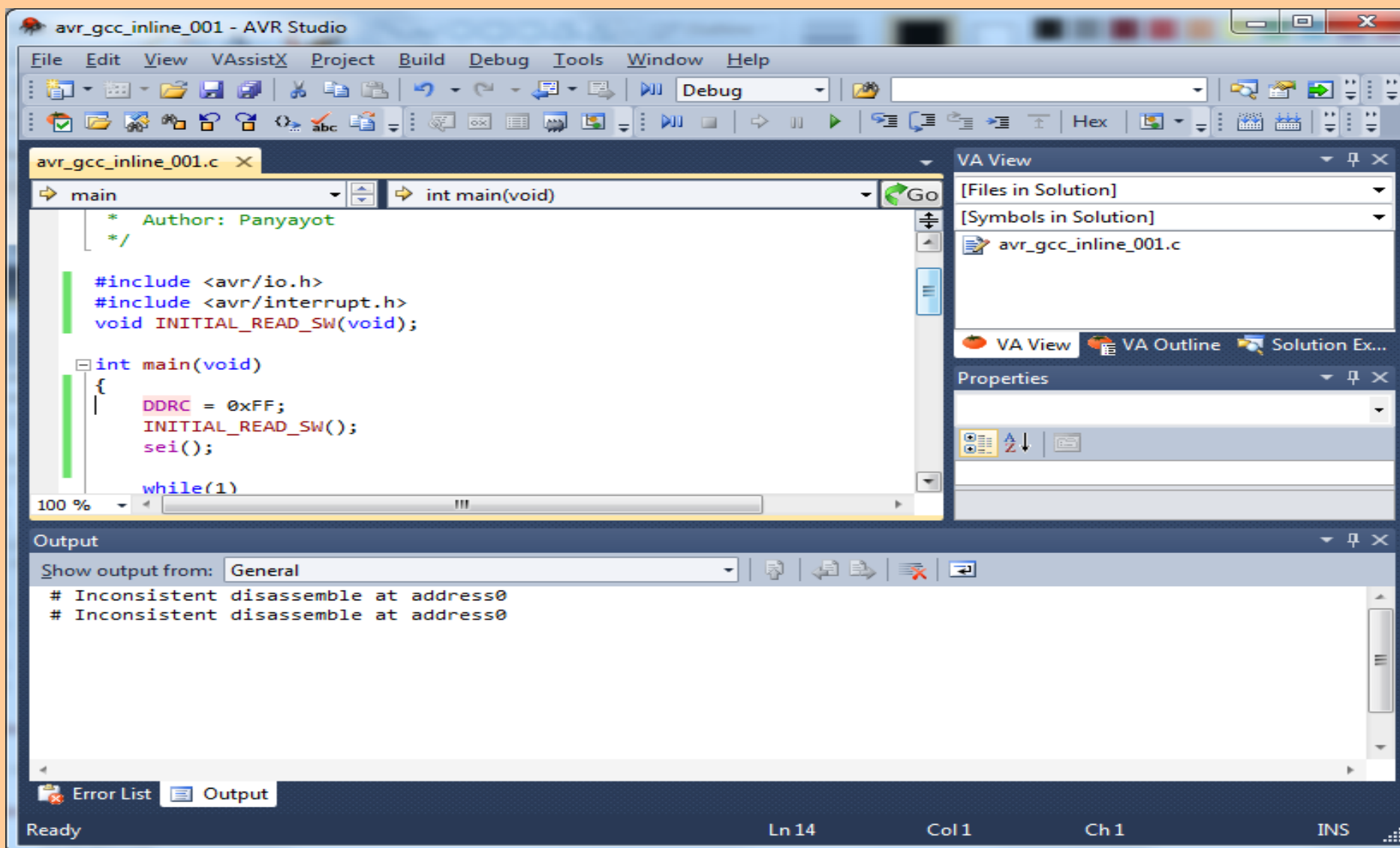
Memory

Program	8/16	abc.	Address: 0x00
000000	0C 94 2A 00 0C 94 47 00	..*..G.	
000004	0C 94 47 00 0C 94 47 00	..G..G.	
000008	0C 94 47 00 0C 94 47 00	..G..G.	
00000C	0C 94 47 00 0C 94 47 00	..G..G.	
000010	0C 94 47 00 0C 94 ED 00	..G..i.	
000014	0C 94 47 00 0C 94 47 00	..G..G.	
000018	0C 94 47 00 0C 94 47 00	..G..G.	
00001C	0C 94 47 00 0C 94 47 00	..G..G.	
000020	0C 94 47 00 0C 94 47 00	..G..G.	
000024	0C 94 47 00 0C 94 47 00	..G..G.	
000028	0C 94 47 00 11 24 1F BE	..G..\$.%	
00002C	CF E5 D8 E0 DE BF CD BF	ï¿ðàþ¿í¿	
000030	10 E0 A0 E6 B0 E0 EA E5	.à æ°àêää	
000034	F2 E0 02 C0 05 90 0D 92	òà.À..'	
000038	A2 37 B1 07 D9 F7 10 E0	ø7±.Û÷.à	
00003C	A2 E7 B0 E0 01 C0 1D 92	øç°à.À.'	



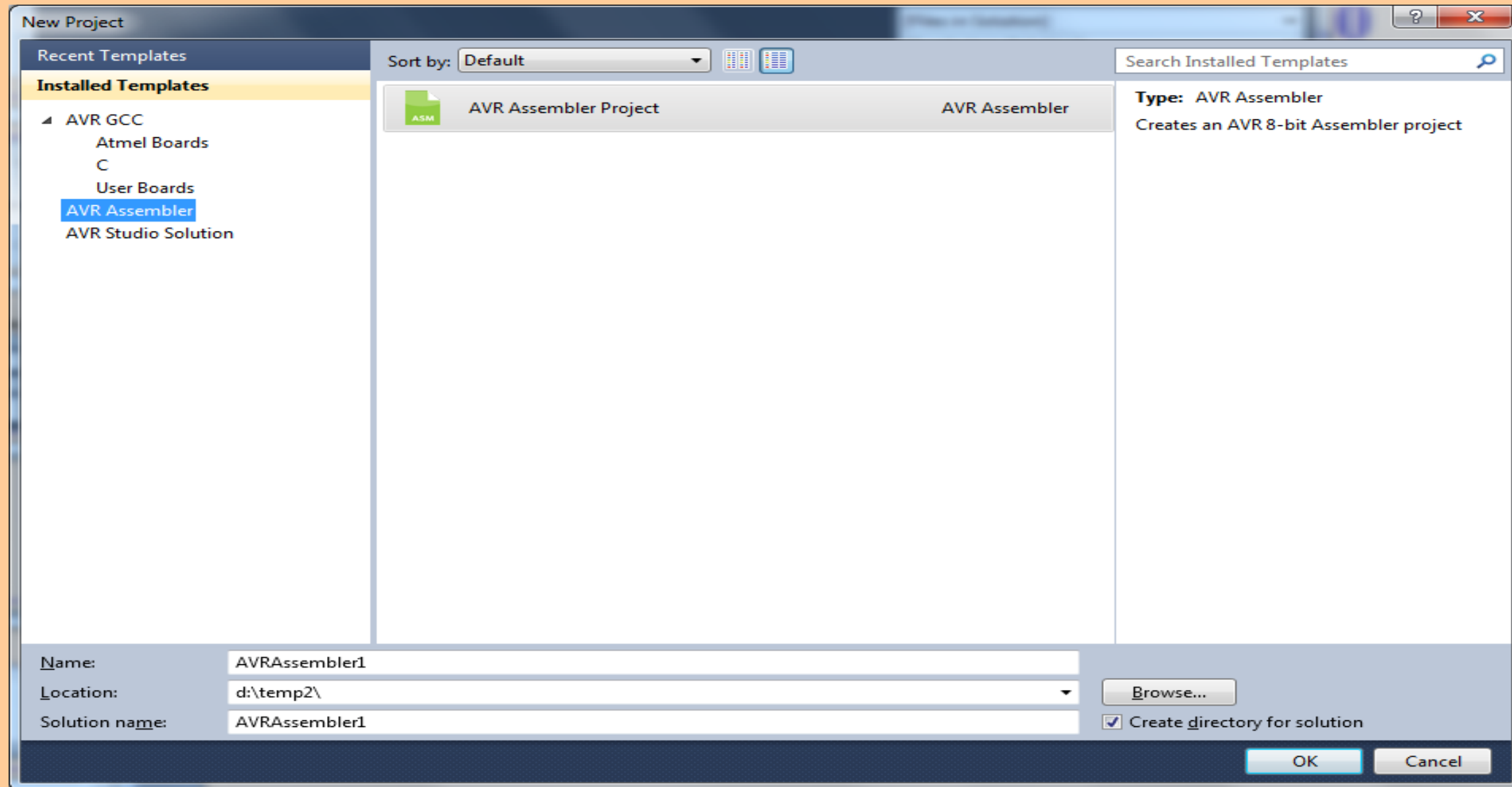
AVR-Studio V5.0 ขึ้นไป

◆ มีทั้ง Assembler และ C Compiler ให้ครบในตัว



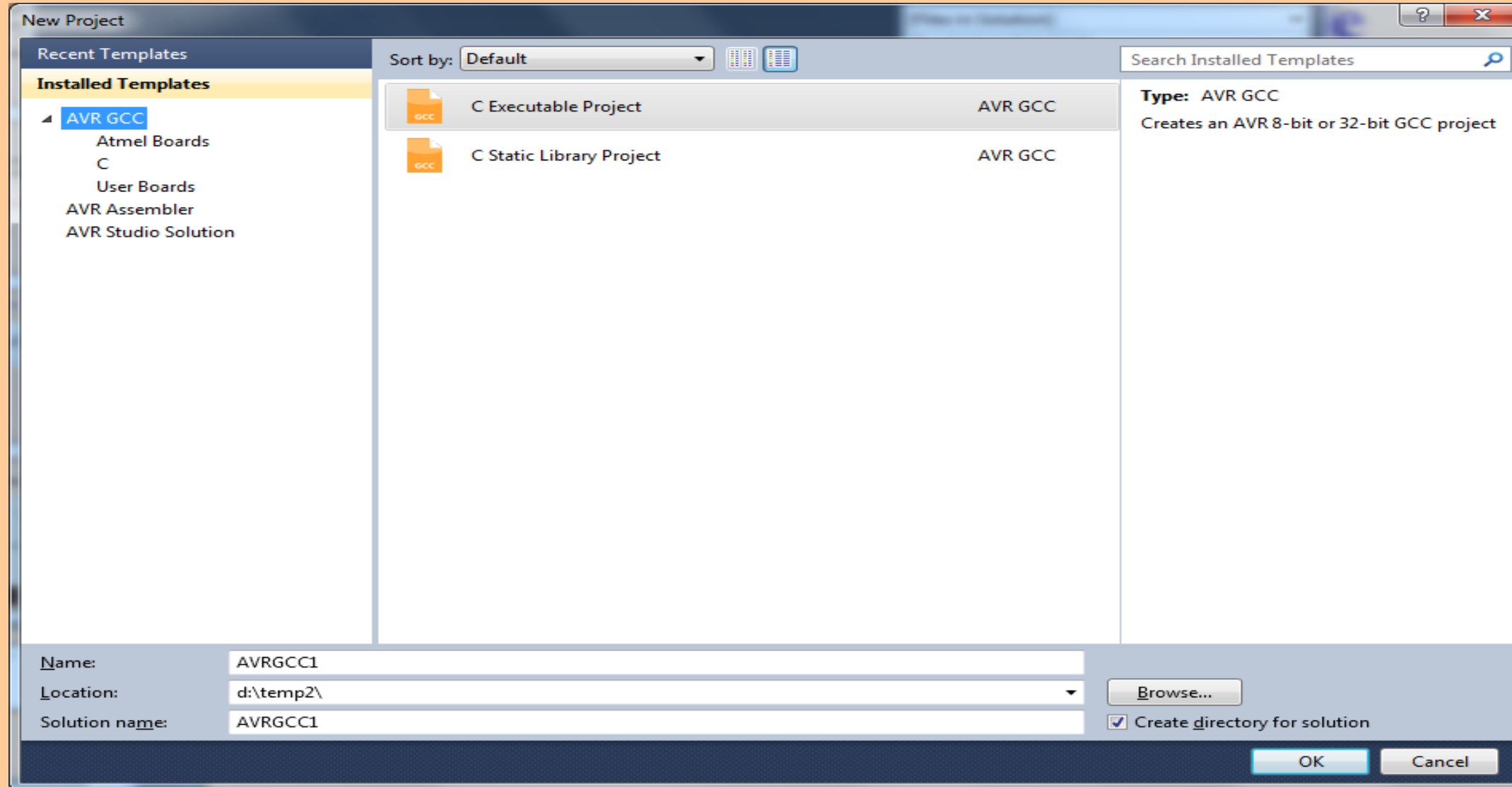
AVR-Studio V5.0

◆ สร้างโปรเจกต์ด้วยภาษาแอสเซมบลี

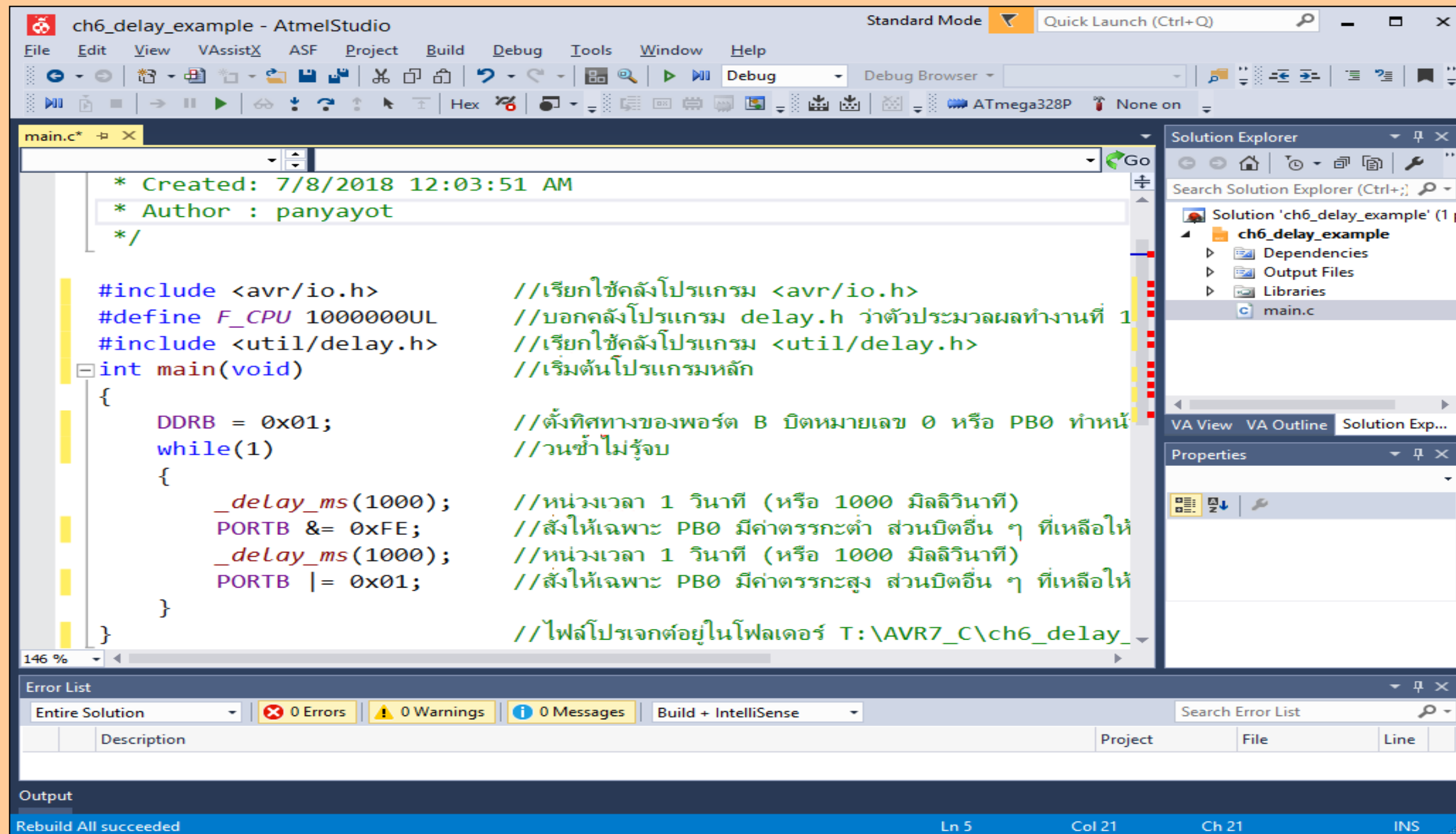


AVR-Studio V5.0

♦ สร้างโปรเจกต์ด้วยภาษาซี

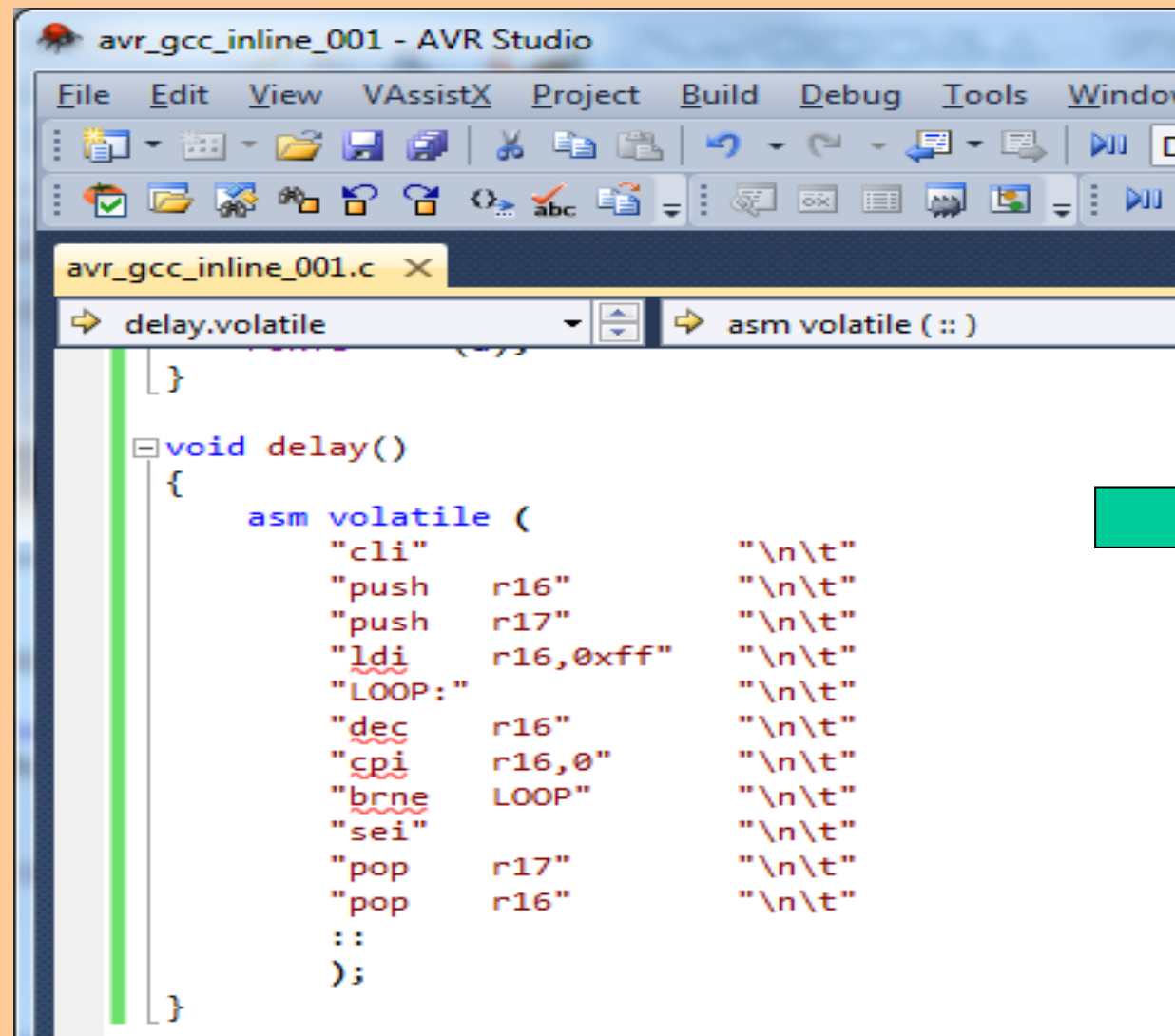


AtmelStudio 7.0



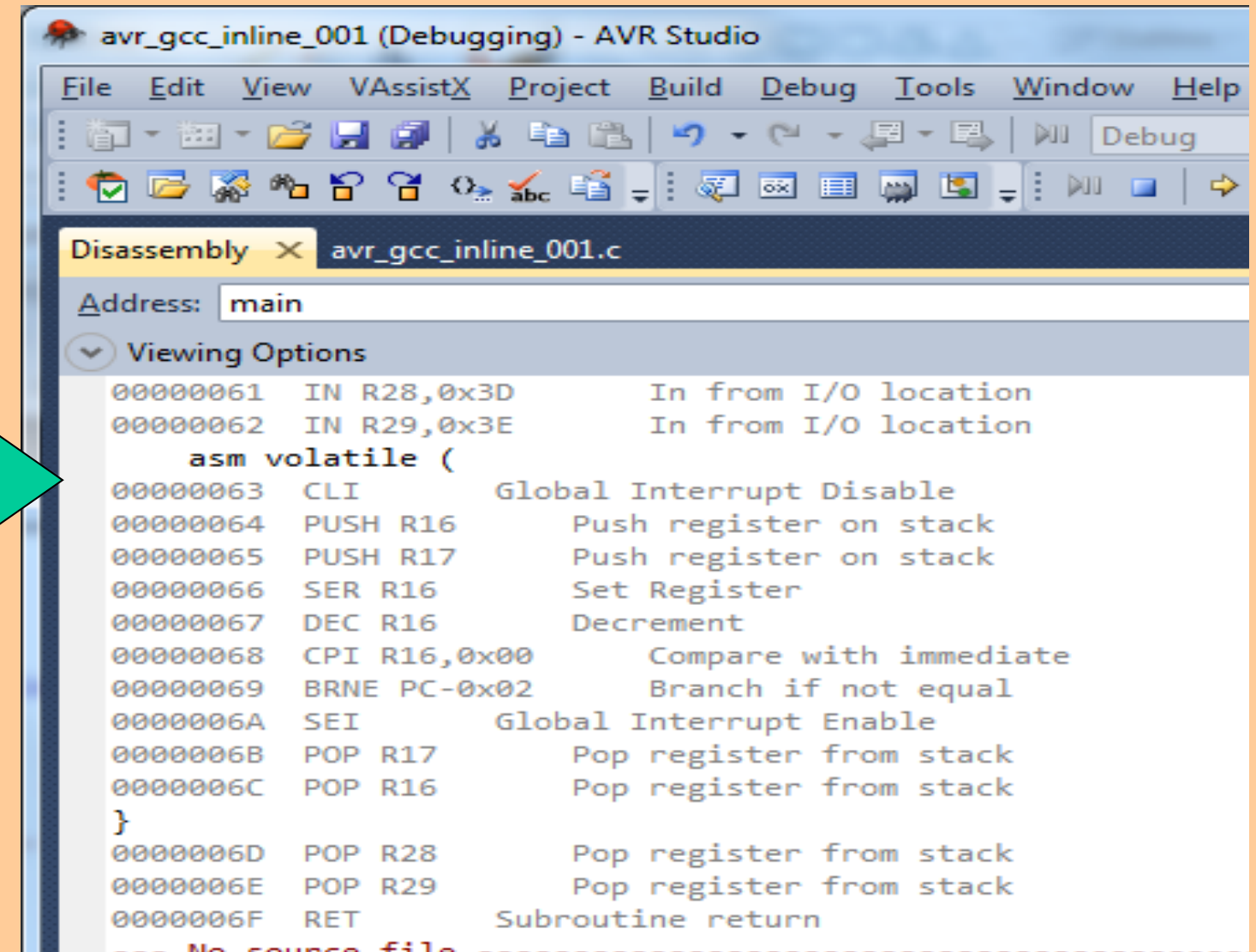
Inline Assembler

◆ เราสามารถแทรกโค้ดภาษาแอสเซมบลีในภาษาซีได้



The screenshot shows the AVR Studio IDE with a C file named `avr_gcc_inline_001.c` open. The code defines a `void delay()` function. Inside the function, there is an `asm volatile (` block containing assembly instructions: `"cli"`, `"push r16"`, `"push r17"`, `"ldi r16,0xff"`, `"LOOP:"`, `"dec r16"`, `"cpi r16,0"`, `"brne LOOP"`, `"sei"`, `"pop r17"`, `"pop r16"`, and `::`. The code ends with `);`. A green arrow points from this code block to the disassembly window on the right.

```
avr_gcc_inline_001.c  
delay	volatile  
asm volatile (::)  
}  
void delay()  
{  
    asm volatile (  
        "cli"                "\n\t"  
        "push r16"           "\n\t"  
        "push r17"           "\n\t"  
        "ldi r16,0xff"        "\n\t"  
        "LOOP:"              "\n\t"  
        "dec r16"             "\n\t"  
        "cpi r16,0"           "\n\t"  
        "brne LOOP"          "\n\t"  
        "sei"                 "\n\t"  
        "pop r17"             "\n\t"  
        "pop r16"             "\n\t"  
        "::  
    );  
}
```



The screenshot shows the AVR Studio IDE in debug mode, displaying the disassembly of the `delay` function. The address range is from `00000061` to `0000006F`. The instructions are: `IN R28,0x3D`, `IN R29,0x3E`, `CLI` (Global Interrupt Disable), `PUSH R16` (Push register on stack), `PUSH R17` (Push register on stack), `SER R16` (Set Register), `DEC R16` (Decrement), `CPI R16,0x00` (Compare with immediate), `BRNE PC-0x02` (Branch if not equal), `SEI` (Global Interrupt Enable), `POP R17` (Pop register from stack), `POP R16` (Pop register from stack), `POP R28` (Pop register from stack), `POP R29` (Pop register from stack), and `RET` (Subroutine return).

```
avr_gcc_inline_001 (Debugging) - AVR Studio  
Disassembly X avr_gcc_inline_001.c  
Address: main  
Viewing Options  
00000061 IN R28,0x3D In from I/O location  
00000062 IN R29,0x3E In from I/O location  
asm volatile (  
00000063 CLI Global Interrupt Disable  
00000064 PUSH R16 Push register on stack  
00000065 PUSH R17 Push register on stack  
00000066 SER R16 Set Register  
00000067 DEC R16 Decrement  
00000068 CPI R16,0x00 Compare with immediate  
00000069 BRNE PC-0x02 Branch if not equal  
0000006A SEI Global Interrupt Enable  
0000006B POP R17 Pop register from stack  
0000006C POP R16 Pop register from stack  
}  
0000006D POP R28 Pop register from stack  
0000006E POP R29 Pop register from stack  
0000006F RET Subroutine return  
--- No source file ---
```





Inline Assembler

◆ ศึกษาเพิ่มเติมได้ที่

GCC-AVR Inline Assembler Cookbook

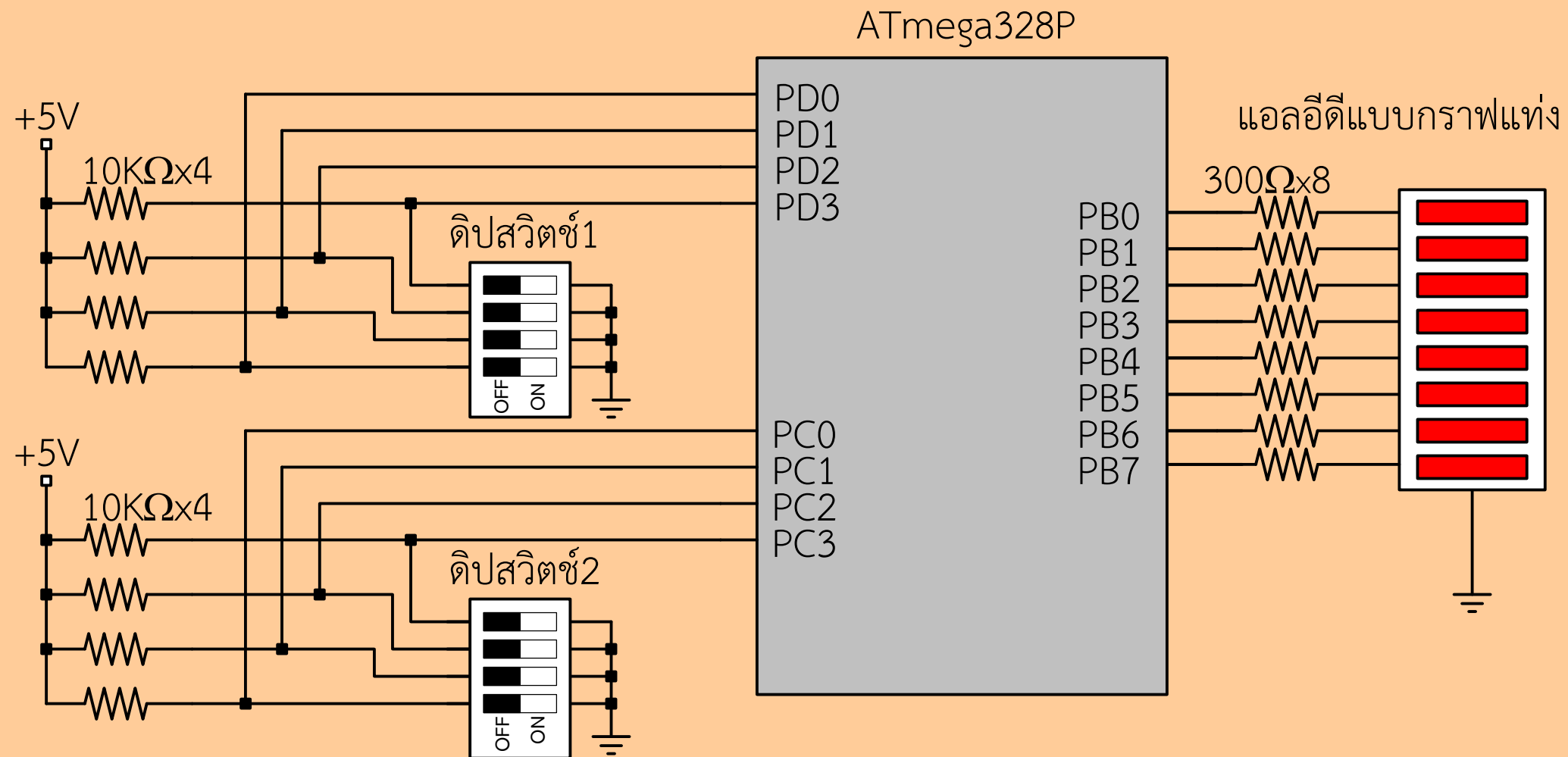
◆ ดาวน์โฮลดคู่มือได้ที่

https://lms2.psu.ac.th/pluginfile.php/542207/mod_resource/content/0/GCCAVRInlAsmCB.pdf



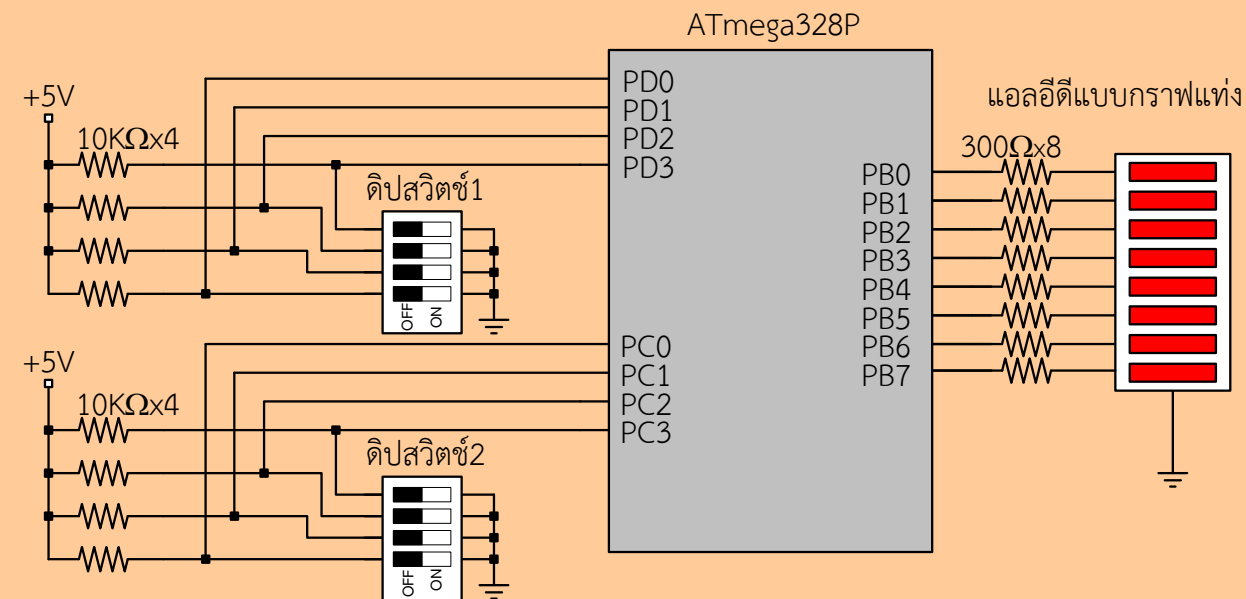


Example 2





Example 2



ดิปสวิตช์1	ดิปสวิตช์2	ค่าที่ต้องแสดงผลทางพอร์ต B	หมายเหตุ
0000_2	1111_2	$0000\ 0000_2$	แอลอีดีดับทุกดวง
0101_2	1010_2	$0011\ 0010_2$	$5*10 = 50 = 110010_2$
0100_2	0011_2	$0000\ 1100_2$	$4*3 = 12 = 1100_2$
1001_2	1000_2	$0100\ 1000_2$	$9*8 = 72 = 100\ 1000_2$
1111_2	1111_2	$1110\ 0001_2$	$15*15 = 225 = 1110\ 0001_2$

Example 2

1	#include <avr/io.h>	//อ่านค่าจากคลังโปรแกรมซึ่งเก็บในแฟ้ม io.h
2	int main(void)	//ฟังก์ชันหลักของโปรแกรม
3	{	//เริ่มต้นฟังก์ชันหลักของโปรแกรม
4	uint8_t d, c, b;	//ประกาศตัวแปร
5	DDRD = 0xF0;	//ตั้งค่าทิศทางของพอร์ต D สี่บิตล่างให้ทำหน้าที่รับเข้า
6	DDRC = 0xF0;	//ตั้งค่าทิศทางของพอร์ต C สี่บิตล่างให้ทำหน้าที่รับเข้า
7	DDRB = 0xFF;	//ตั้งค่าทิศทางของพอร์ต B ให้ทำหน้าที่ส่งออก
8	while (1)	//วนซ้ำแบบไม่รู้จบ
9	{	//เริ่มต้นขอบเขตของการวนซ้ำแบบไม่รู้จบ
10	d = PIND;	//อ่านค่าจากสวิตช์ที่ต่ออยู่กับพอร์ต D
11	c = PINC;	//อ่านค่าจากสวิตช์ที่ต่ออยู่กับพอร์ต C
12	d &= 0x0F;	//พรางสี่บิตบนของตัวแปร d ทิ้งไป
13	c &= 0x0F;	//พรางสี่บิตบนของตัวแปร c ทิ้งไป
14	b = d * c;	//คำนวณ $b \leftarrow d * c$
15	PORTB = b;	//ส่งผลลัพธ์ในการคำนวณออกแสดงผล ณ พอร์ต B
16	}	//สิ้นสุดขอบเขตของการวนซ้ำแบบไม่รู้จบ
17	}	//สิ้นสุดฟังก์ชันหลักของโปรแกรม



ฟังก์ชันสำหรับการหน่วงเวลาบน Microchip Studio

```
#define F_CPU 1000000UL // 1 MHz
```

```
#include <util/delay.h>
```

```
void _delay_ms (double __ms)
```

```
void _delay_us (double __us)
```

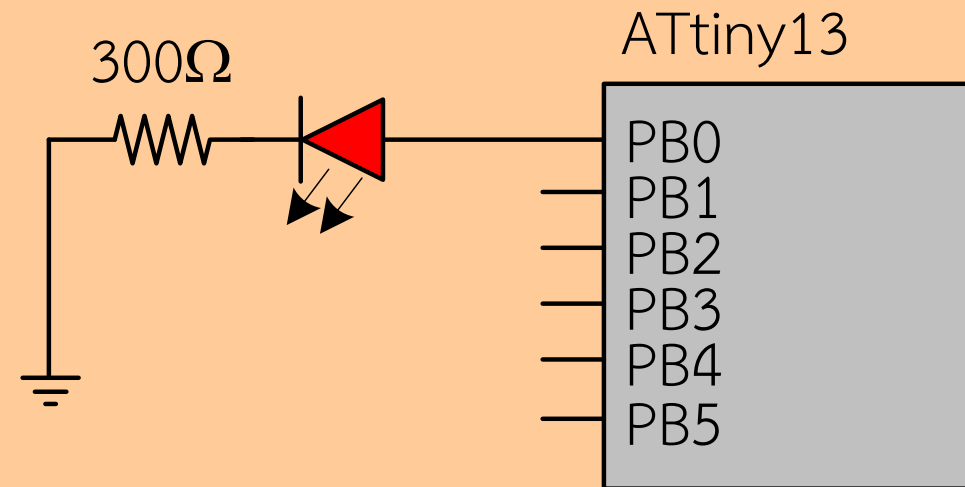
◆ ตัวอย่างการตั้งค่าความถี่ตัวประมวลผลที่ 12 MHz

◆ #define F_CPU 12000000UL //12 MHz





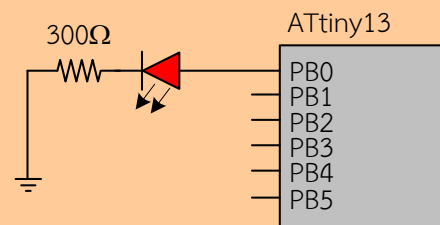
Example 3



จงเขียนโปรแกรมเพื่อให้แอลอีดีกระพริบติดสลับกับดับทุก ๆ 1 วินาที



Example 3



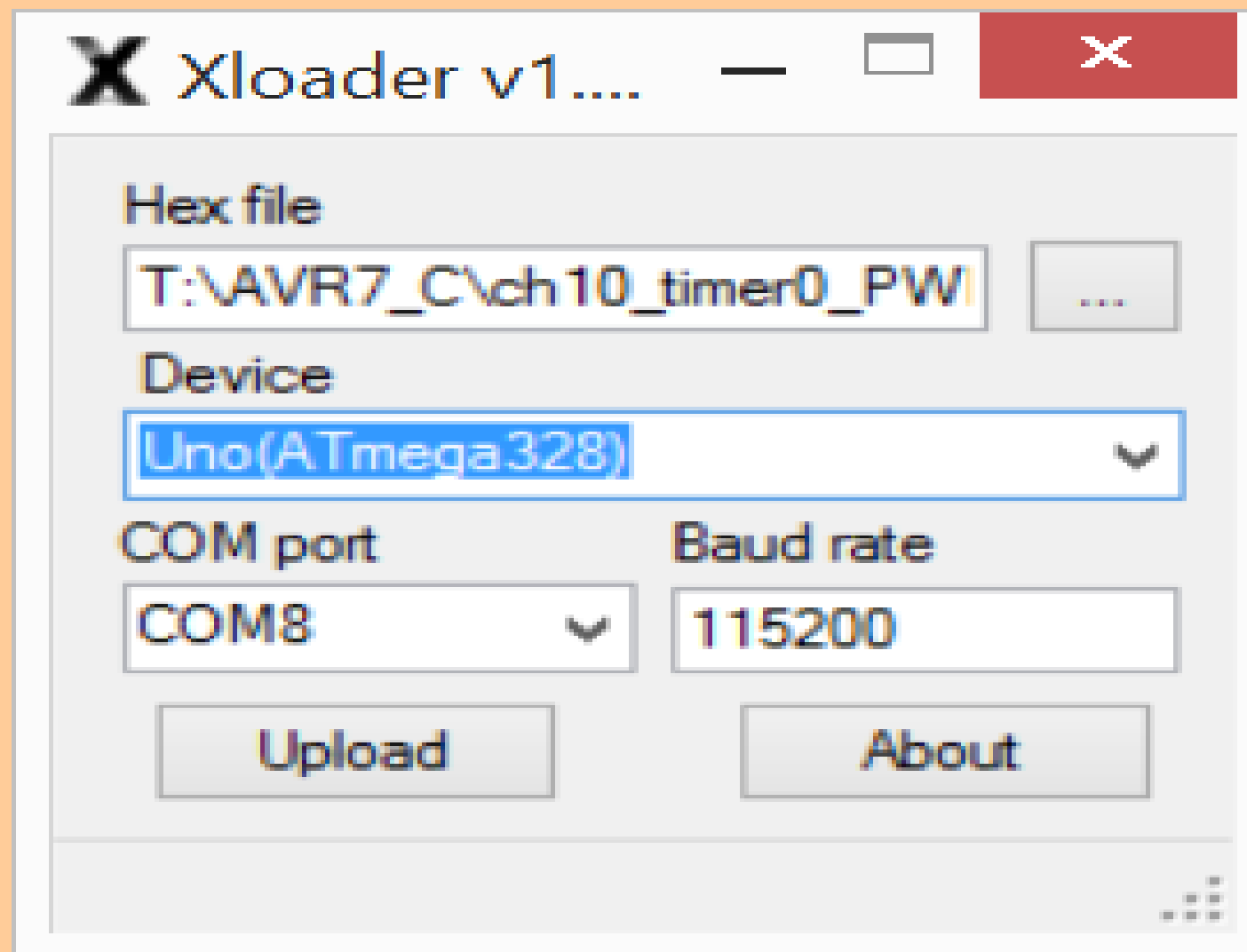
```
1  #include <avr/io.h>           //เรียกใช้คลังโปรแกรม <avr/io.h>
2  #define F_CPU 1.2E6           //บอกคลังโปรแกรม delay.h ว่าตัวประมวลผลทำงานที่ 1.2 MHz
3  #include <util/delay.h>       //เรียกใช้คลังโปรแกรม <util/delay.h>
4  int main(void)                //เริ่มต้นโปรแกรมหลัก
5  {
6      DDRB = 0x01;              //ตั้งทิศทางของพอร์ต B บิตหมายเลข 0 หรือ PB0 ทำหน้าที่ส่งออก
7      while(1)                  //วนซ้ำไม่รู้จบ
8      {
9          _delay_ms(1000);       //หน่วงเวลา 1 วินาที (หรือ 1000 มิลลิวินาที)
10         PORTB &= 0xFE;         //สั่งให้เฉพาะ PB0 มีค่าตรรกะต่ำ ส่วนบิตอื่น ๆ ที่เหลือให้คงค่าเดิม
11         _delay_ms(1000);       //หน่วงเวลา 1 วินาที (หรือ 1000 มิลลิวินาที)
12         PORTB |= 0x01;         //สั่งให้เฉพาะ PB0 มีค่าตรรกะสูง ส่วนบิตอื่น ๆ ที่เหลือให้คงค่าเดิม
13     }
14 }
```



การบันทึกโปรแกรมสู่ไมโครคอนโทรลเลอร์

◆ ใช้ Arduino IDE

◆ ใช้ Xloader





ฉบับที่ 1

