

Computer Programming II

การเขียนโปรแกรมคอมพิวเตอร์2

LECTURE#3 Operator & Expression

อ.สฤติย์ ประสมพันธ์

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ

KMUTNB

คำสั่งและนิพจน์ (Statement and Expression)

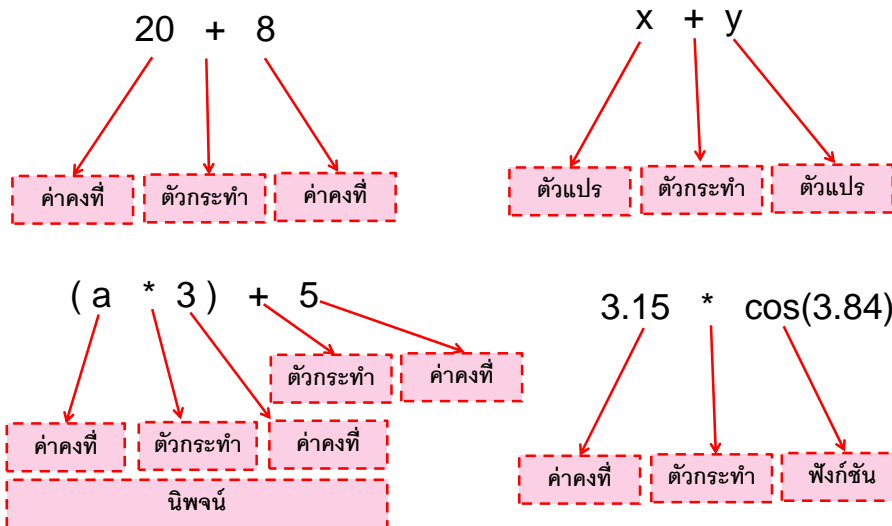
คำสั่ง, ข้อคำสั่ง (Statement) คือขั้นตอนในการทำงานหนึ่งขั้นตอน
ทุกคำสั่งต้องจบด้วยเครื่องหมาย ;

กลุ่มคำสั่ง คือคำสั่งที่อยู่ในวงเล็บปีกกา {}

นิพจน์ (Expression) คือการกระทำเพื่อให้ได้ผลลัพธ์ค่าหนึ่งค่า
ประกอบไปด้วยตัวถูกกระทำ (Operands) และตัวกระทำ
(Operators) เขียนเรียงกันไป

เช่น $3 * 2 - 1 + 7$ หรือ $a * 5$ เป็นต้น

ตัวอย่างการเขียนนิพจน์คณิตศาสตร์และนิพจน์ภาษา C



นิพจน์

- **นิพจน์คณิตศาสตร์ (Arithmetic Expression)** หมายถึง การนำตัวแปรค่าคงที่มาสัมพันธ์กันโดยใช้เครื่องหมายคณิตศาสตร์เป็นตัวเชื่อมผลที่ได้จากนิพจน์แบบนี้จะเป็นตัวเลข
- **นิพจน์ตรรกะ (Logical Expression)** หมายถึงการนำตัวแปรค่าคงที่หรือนิพจน์มาสัมพันธ์กันโดยใช้เครื่องหมายเปรียบเทียบและเครื่องหมายตรรกะเป็นตัวเชื่อม ผลที่ได้จะออกมาเป็นจริงหรือเท็จ

กฎเกณฑ์การเขียนนิพจน์

- 1. ห้ามเขียนตัวแปร 2 ตัวติดกันโดยไม่มีเครื่องหมายเช่น ab
- 2. ถ้าเขียนนิพจน์โดยมีค่าของตัวแปร หรือค่าคงที่ต่างชนิดกันในนิพจน์เดียวกันภาษาซีจะเปลี่ยนชนิดของข้อมูลที่มีขนาดเล็กให้เป็นชนิดข้อมูลที่มีขนาดใหญ่ขึ้น

เครื่องหมายดำเนินการ (Operators)

- กำหนดการกระทำที่เกิดขึ้นกับตัวแปรและค่าคงที่ โดยที่นิพจน์ประกอบด้วยตัวแปร และค่าคงที่ และใช้ตัวดำเนินการคำนวณเพื่อให้ได้ค่า ได้แก่
 - ตัวดำเนินการทางคณิตศาสตร์
 - ตัวดำเนินการสัมพันธ์และตัวดำเนินการตรรกะ
 - ตัวดำเนินการประกอบ
 - ตัวดำเนินการระดับบิต
 - ตัวดำเนินการบอกชนิดตัวแปร (type) และตัวดำเนินการบอกขนาดหน่วยความจำของตัวแปร (size of) เป็นต้น

ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

เครื่องหมาย	ความหมาย	ตัวอย่าง
+	การบวก	A+B
-	การลบ	A-B
*	การคูณ	A*B
/	การหาร	A/B
%	การหารเอาแต่เศษไว้ (Modulo)	5%3=1 เศษ 2 จะเก็บแต่เศษ 2 เอาไว้
--	การลดค่าลงครั้งละ 1	A -- จะเหมือนกับ A=A-1
++	การเพิ่มค่าขึ้นครั้งละ 1	A++ จะเหมือนกับ A=A+1

การดำเนินการแบบ Binary

$$a = 4$$

$$4 + 0 + 1 = 5$$

$$a++ + 1 = 4$$

การดำเนินการแบบ Unary

เครื่องหมาย	ความหมาย	ตัวอย่าง
+	เป็นตัวดำเนินการที่ระบุค่าบวกสำหรับ Operand	c 4+a +=
-	เป็นตัวดำเนินการที่ระบุค่าลบสำหรับ Operand	c #a -=
++	การเพิ่มค่าให้ Operand 1 ค่า	++a
--	การลดค่าให้ Operand 1 ค่า	--a

- Prefix unary operator จะคำนวณ unary operator ก่อน binary operator
- Postfix unary operator จะคำนวณ binary operator ก่อน

การดำเนินการแบบ Unary

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

```
void main()
```

```
{
```

```
    int i, j, N;
```

```
    i=10; j= 2; N=i* ++j;
```

```
    printf("N=i* ++j=%d, i=%d, j=%d\n", N, i, j);
```

```
    i=10; j= 2; N=i* j++;
```

```
    printf("N=i* j++=%d, i=%d, j=%d\n", N, i, j);
```

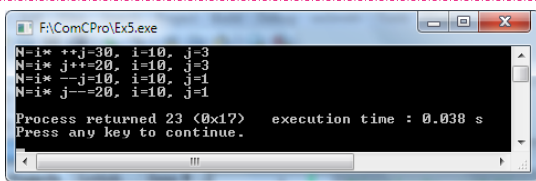
```
    i=10; j= 2; N=i* --j;
```

```
    printf("N=i* --j=%d, i=%d, j=%d\n", N, i, j);
```

```
    i=10; j= 2; N=i*j--;
```

```
    printf("N=i* j--=%d, i=%d, j=%d\n", N, i, j);
```

```
}
```



การทำงานของตัวดำเนินการ

- การโอเปอเรชัน (Operation) หรือการทำงานของตัวดำเนินการต้องเป็นไปตามลำดับของการวางค่าคงที่หรือตัวแปรที่จะกระทำต่อกัน ซึ่งเรียกว่า **ลำดับของการโอเปอเรชัน** ซึ่งลำดับการทำงานของโอเปอเรเตอร์โดยคอมไพเลอร์ภาษา C ต้องเป็นไปตาม

กฎการกำหนดลำดับการทำงานของตัวดำเนินการ (Precedence)

ลำดับการทำงานก่อน-หลังของตัวดำเนินการ

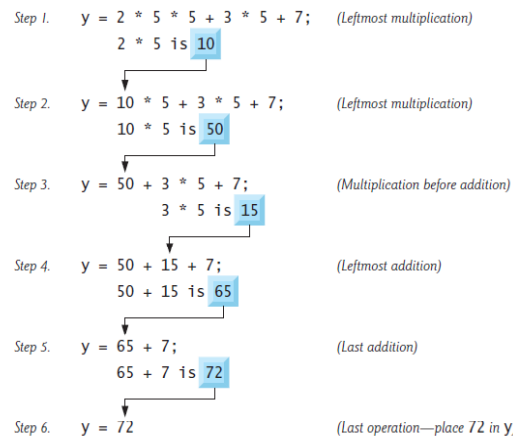
ตัวดำเนินการ	ทิศทางการดำเนินการ
0, [], ->, , ~, ++, --, +(ค่าบวก), -(ค่าลบ), *, &(type), sizeof	ซ้ายไปขวา
*, /, %	ซ้ายไปขวา
+, - (ตัวกระทำทางคณิตศาสตร์)	ซ้ายไปขวา
<<, >>	ซ้ายไปขวา
<, <=, >, >=	ซ้ายไปขวา
==, !=	ซ้ายไปขวา
&	ซ้ายไปขวา
^	ซ้ายไปขวา
	ซ้ายไปขวา
&& != &	ซ้ายไปขวา
	ซ้ายไปขวา
?:	ขวาไปซ้าย
=, +=, -=, /=, %=, &=, ^=, =, <<=, >>=	ขวาไปซ้าย
,	ซ้ายไปขวา

กฎการกำหนดลำดับการทำงานของตัวดำเนินการ (Precedence)

โอเปอเรเตอร์	ลำดับของการโอเปอเรชัน	
()	ซ้ายไปขวา	$z = pr \% q + w / x - y$ $z = p * r \% q + w / x - y;$ <div>6 1 2 4 3 5</div>
-	ซ้ายไปขวา	
แสดงความเป็น ลบของตัวเลข		$z = a * b + c / d - e$ <div>5 1 3 2 4</div>
* /	ซ้ายไปขวา	
+ -	ซ้ายไปขวา	

การกำหนดลำดับการทำงานของตัวดำเนินการ (Precedence)

$$y = a * x * x + b * x + c;$$



การทำงานของตัวดำเนินการ

- การคำนวณ $a + b * c$
ภาษา C จะกระทำเครื่องหมาย * ก่อนเครื่องหมาย + แต่ถ้าเครื่องหมายมีความเท่าเทียมกันจะกระทำจากซ้ายไปขวา เช่น $a+b+c$ จะเริ่มทำจาก a บวก b แล้วจึงนำค่าที่ได้มาบวก c ถ้าต้องการกำหนดลำดับให้ทำโดยการใส่เครื่องหมายวงเล็บในส่วนที่ต้องการให้ทำงานก่อน
- ตัวดำเนินการ %
เป็นตัวดำเนินการทางคณิตศาสตร์เพียงตัวเดียวที่กำหนดให้ใช้กับค่าจำนวนเต็ม เท่านั้น นอกนั้นสามารถใช้กับค่าอื่นๆ ได้
- เครื่องหมาย /
 a / b ถ้าทั้งตัวแปร a และ b เป็น integer ค่าที่ได้จากการดำเนินการของหารจะมีชนิดเป็น integer ถ้า a หรือ b ตัวแปรใดตัวแปรหนึ่งเป็นตัวแปรชนิด float จะได้คำตอบเป็นชนิด float เช่น
 $39 / 5$ จะมีค่าเท่ากับ 7
 $39.0 / 5$ จะมีค่าเท่ากับ 7.8

ตัวดำเนินการเพิ่ม และลดค่า (Increment & Decrement)

- ตัวดำเนินการเพิ่ม และลดค่า เป็นตัวดำเนินการในการเพิ่มหรือลดค่าของตัวแปร โดย
 - ตัวดำเนินการเพิ่มค่า ++ จะบวกหนึ่งเข้ากับตัวถูกดำเนินการ
 - ตัวดำเนินการลดค่า -- จะลบหนึ่งออกจากตัวถูกดำเนินการ
- ตัวดำเนินการเพิ่ม และลดค่ามีวิธีใช้งาน 2 แบบคือ
 - ใช้เป็นตัวดำเนินการแบบมาก่อน (prefix) เช่น ++n
 - ใช้ตัวดำเนินการแบบตามหลัง (postfix) ก็ได้เช่น n++
- ในทั้งสองกรณี ผลลัพธ์คือการเพิ่มค่า 1 ให้กับ n
 - แต่นิพจน์ ++n จะเพิ่มค่าก่อนที่จะนำค่าไปใช้
 - ขณะที่ n++ จะเพิ่มค่าหลังจากนำค่าไปใช้ ซึ่งหมายความว่าถ้ามีการนำค่าไปใช้ (ไม่เพียงหวังเฉพาะผลลัพธ์) ++n และ n++ จะแตกต่างกัน

ตัวดำเนินการเพิ่ม และลดค่า (Increment & Decrement)

ตัวอย่าง

ถ้า n มีค่า 5

$x = n++;$ จะเป็นการกำหนดค่า 5 ให้กับ x

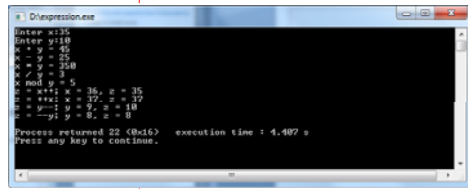
$x = ++n;$ จะเป็นการกำหนดค่า 6 ให้กับ x

ตัวดำเนินการนี้จะใช้ได้กับเฉพาะตัวแปรเท่านั้น จะใช้กับนิพจน์อื่นๆ

ไม่ได้เช่น $(i+j)++$

การทำงานของตัวดำเนินการ

```
#include<stdio.h>
main ( )
{
    int x,y,z;
    printf("Enter x:");
    scanf("%d",&x);
    printf("Enter y:");
    scanf("%d",&y);
    printf("x + y = %d\n", x+y);
    printf("x - y = %d\n", x-y);
    printf("x * y = %d\n", x*y);
    printf("x / y = %d\n", x/y);
    printf("x mod y = %d\n", x%y);
    z=x++;
    printf("z = x++; x = %d, z = %d\n", x,z);
    z=++x;
    printf("z = ++x; x = %d, z = %d\n", x,z);
    z=y--;
    printf("z = y--; y = %d, z = %d\n", y,z);
    z=-y;
    printf("z = -y; y = %d, z = %d\n", y,z);
}
```



ตัวดำเนินการสัมพันธ์และตัวดำเนินการตรรกะ (Relational, Equality, and Logical Operators)

Operator	เครื่องหมาย	ความหมาย
ตัวดำเนินการสัมพันธ์ (Relational Operator)	<	น้อยกว่า
	>	มากกว่า
	<=	น้อยกว่า หรือ เท่ากับ
	>=	มากกว่า หรือ เท่ากับ
ตัวดำเนินการเท่ากับ (Equality Operator)	==	เท่ากับ
	!=	ไม่เท่ากับ
ตัวดำเนินการตรรกะ (Logical Operator)	!	นิเสธ
	&&	และ
		หรือ

ตัวดำเนินการสัมพันธ์ (Relational Operator)

- ตัวดำเนินการสัมพันธ์ เป็นเครื่องหมายที่ใช้ในการเปรียบเทียบและตัดสินใจ ซึ่งผลของการเปรียบเทียบจะเป็นได้ 2 กรณีเท่านั้นคือ จริง และเท็จ
- ค่าทางตรรกะจริงและเท็จมีชนิดเป็น int ดังนั้นผลการกระทำทางตรรกะจึงมีค่าเป็นจำนวนเต็ม และมีค่าได้เพียงสองค่าคือ 1 หรือตัวเลขใดๆ แทนค่าความจริงเป็นจริง และ 0 แทนค่าความจริงเป็นเท็จ (0 เป็นเท็จ ส่วนตัวเลขอื่นๆ ทั้งหมดมีค่าเป็นจริง) เครื่องหมายที่เป็นตัวดำเนินการสัมพันธ์มี 4 ตัวคือ

< (น้อยกว่า) > (มากกว่า) <= (น้อยกว่าเท่ากับ) และ >= (มากกว่าเท่ากับ)

ตัวดำเนินการเท่ากับ (Equality Operator)

ใช้ในการเปรียบเทียบค่า 2 ค่า ว่ามีค่าเท่าหรือไม่เท่ากัน ผลลัพธ์ที่ได้ จะมีเพียง 2 ค่าคือ จริง และ เท็จ เครื่องหมายที่เป็นตัวดำเนินการเท่ากับมี 2 ตัวคือ

== (เท่ากับ) และ != (ไม่เท่ากับ)

ตัวอย่างการใช้งาน	คำอธิบาย
c == 'A'	/* ถ้าตัวแปร c เก็บค่าอักขระ A ผลลัพธ์จะได้ค่าจริง */
k != -2	/* ถ้าตัวแปร k เก็บค่าตัวเลข -2 ผลลัพธ์จะได้ค่าเท็จ */
x+y == 2 * z - 5	
a=b ❌	/* การเปรียบเทียบค่าตัวแปร a กับ b ว่ามีค่าเท่ากันหรือไม่ กลายเป็นการให้ค่าตัวแปร a ด้วยค่าตัวแปร b */
a= b-1 ❌	/* ใช้งานผิดรูปแบบ โดยมีช่องว่างระหว่างเครื่องหมาย = ทั้ง 2 ตัว */

ตัวดำเนินการสัมพันธ์ (Relational Operator)

```
#include <stdio.h>
int main( void )
{
    int num1; /* first number to be read from user */
    int num2; /* second number to be read from user */
    printf( "Enter two integers, and I will tell you\n" );
    printf( "the relationships they satisfy: " );
    scanf( "%d%d", &num1, &num2 ); /* read two integers */
    if ( num1 == num2 ) {
        printf( "%d is equal to %d\n", num1, num2 );
    } /* end if */
    if ( num1 != num2 ) {
        printf( "%d is not equal to %d\n", num1, num2 );
    } /* end if */
    if ( num1 < num2 ) {
        printf( "%d is less than %d\n", num1, num2 );
    } /* end if */
    if ( num1 > num2 ) {
        printf( "%d is greater than %d\n", num1, num2 );
    } /* end if */
    if ( num1 <= num2 ) {
        printf( "%d is less than or equal to %d\n", num1, num2 );
    } /* end if */
    if ( num1 >= num2 ) {
        printf( "%d is greater than or equal to %d\n", num1, num2 );
    } /* end if */
    return 0; /* indicate that program ended successfully */
} /* end function main */
```

ตัวดำเนินการตรรกะ (Logical Operator)

ใช้ในการเปรียบเทียบ และกระทำทางตรรกะกับค่าตัวเลข หรือค่าที่อยู่ในตัวแปร ผลลัพธ์ที่ได้ จะมีเพียง 2 ค่าคือ จริง และ เท็จ

เครื่องหมาย		ตัวถูกกระทำ	ตัวอย่างการใช้งานชนิดแบบ	ตัวอย่างการใช้งานที่ถูกต้อง
!(นิเสธ)	การกลับค่าความจริงของค่าตัวเลขหรือตัวแปร	ต้องการตัวถูกกระทำเพียง 1 ตัว	a! a! = b	!a !(x+7.7) !(a<b c<d)
&& (และ)		ต้องการตัวถูกกระทำ 2 ตัว	a && a & b &b	a&&b !(a<b) && C /* นำ a มาเปรียบเทียบกับ b แล้วนำผลลัพธ์มานิเสธ จากนั้นนำ && กับ C */b
(หรือ)		ต้องการตัวถูกกระทำ 2 ตัว	a b	a b

ตัวดำเนินการตรรกะ (Logical Operator)

ตัวดำเนินการ &&	P	Q	P && Q
	0	0	0
	0	1	0
	1	0	0
	1	1	1
ตัวดำเนินการ	P	Q	P Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1
ตัวดำเนินการ !	P	!P	
	0	1	
	1	0	

ตัวดำเนินการประกอบ (Compound Operator)

ตัวดำเนินการประกอบ คือ ตัวดำเนินการที่เป็นรูปแบบย่อของตัวดำเนินการ+ตัวแปรที่ถูกดำเนินการ ดังนี้ += -= /= %= <<= >>= |= ^=

ตัวอย่าง

i = i + 1;	ตัวดำเนินการประกอบคือ	i += 1;
i = i - a;	ตัวดำเนินการประกอบคือ	i -= a;
i = 1 * (a + 1);	ตัวดำเนินการประกอบคือ	i *= a+1;
i = i / (a-b);	ตัวดำเนินการประกอบคือ	i /= a-b;
i = i %101;	ตัวดำเนินการประกอบคือ	i %= 101;
i = i << 1;	ตัวดำเนินการประกอบคือ	i <<= 1;
i = i >> i;	ตัวดำเนินการประกอบคือ	i >>= i;
i = i & 01;	ตัวดำเนินการประกอบคือ	i = 0xf;
i = i & 01;	ตัวดำเนินการประกอบคือ	i = 0xf;
i = i ^ (07 0xb);	ตัวดำเนินการประกอบคือ	i ^= 07 0xb;

ตัวกระทำบอกขนาด (Sizeof Operator)

- เป็นตัวกระทำใช้รายงานขนาดของหน่วยความจำที่ใช้ในการเก็บค่า โดยมีรูปแบบดังนี้

sizeof ค่าคงที่ หรือ sizeof (แบบของตัวแปร)

ตัวอย่าง

```
int ABC;  
printf("%d",sizeof(ABC));
```

ผลลัพธ์

2

จะเห็นว่ามีการพิมพ์ค่าขนาดของตัวแปรชื่อ ABC ที่มีชนิดเป็น integer ออกทางหน้าจอ ค่า 2 คือขนาดของ integer ทั้งหมดนั่นเอง

ตัวดำเนินการแบบมีเงื่อนไข (Conditional Operator)

- ตัวกระทำสำหรับเลือกค่า เป็นตัวกระทำใช้ในการเลือกการให้ค่าของนิพจน์ ตัวกระทำชนิดนี้ประกอบไปด้วยเครื่องหมาย ? และ นิพจน์เลือกค่า (Condition Expressions) จะเขียนอยู่ในรูป

นิพจน์1 ? นิพจน์2 : นิพจน์3

นิพจน์เลือกค่ามีการให้ค่าดังนี้คือ หากค่าในนิพจน์1 มีค่าไม่เท่ากับ 0 (เป็นจริง) จะให้ค่านิพจน์เป็นไปตามนิพจน์ที่ 2 แต่ถ้าค่าในนิพจน์1 เป็น 0 จะให้ค่านิพจน์เป็นไปตามนิพจน์ที่ 3 เช่น

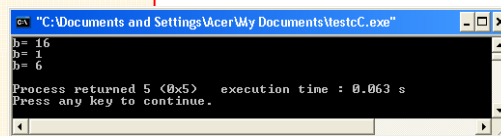
$c = ((a+5)? (a+1) : 0);$

สมมติให้ค่า a เป็น 1 ซึ่งทำให้นิพจน์ a+5 เป็นจริง ก็จะทำให้ค่า c เป็น 2 ซึ่งได้จาก a+1 และถ้าสมมติให้ a เป็น -5 ก็ทำให้นิพจน์ a+5 เป็นเท็จ c ก็จะได้รับค่า 0 ดังตัวอย่างต่อไปนี้

$$\begin{aligned} & \text{if}(c \neq 0) \\ & \text{then } c = a + 1 \\ & \text{else } c = 0 \end{aligned}$$

ตัวดำเนินการแบบมีเงื่อนไข (Conditional Operator)

```
#include<stdio.h>  
int main(void)  
{  
    int a,b;  
    a = 15;  
    b = ((a+5)? (a+1) : 0);  
    printf("b= %d\n",b);  
    a=0;  
    b=((a+5)?(a+1) : 0);  
    printf("b= %d\n",b);  
    a=5;  
    b=((a+5)?(a+1) : 0);  
    printf("b= %d\n",b);  
}
```



ถ้า a เป็น 15
%d แทนค่าของ a = 0

การแปลงค่าผลลัพธ์เป็นตัวแปรชนิดใหม่ (Casting)

- ผลของการกระทำของนิพจน์จะให้ค่าออกมาค่าหนึ่งเสมอ ค่าที่ได้จะมีชนิดสอดคล้อง กับตัวกระทำ และตัวถูกกระทำภายในนิพจน์นั้น ๆ เช่น ตัวถูกกระทำเป็น int ค่าที่ได้จะเป็นชนิด int ด้วย
- เราอาจเปลี่ยนชนิดของค่านั้น ๆ ให้มีชนิดตามที่เรต้องการได้โดยการเขียนชนิดของข้อมูลแบบใหม่ ภายในวงเล็บ นำหน้านิพจน์นั้น ๆ

(แบบข้อมูลแบบใหม่) นิพจน์

การแปลงค่าผลลัพธ์เป็นตัวแปรชนิดใหม่ (Casting)

- ตัวอย่าง

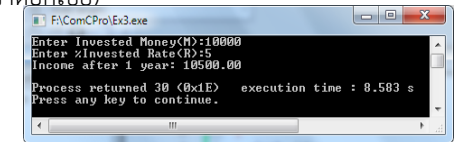
$(int)(a*5.2)$ เป็นการเปลี่ยนค่า output เป็นข้อมูลชนิด int
 $(float)(b+5)$ เป็นการเปลี่ยนค่า output เป็นข้อมูลชนิด float

การแปลงค่าผลลัพธ์เป็นตัวแปรชนิดใหม่ (Casting)

- ตัวอย่าง เขียนโปรแกรมสำหรับการคำนวณเงินฝากพร้อมดอกเบี้ยเมื่อเวลาผ่านไป 1 ปี ที่คำนวณเงินได้แบบดอกเบี้ยทบต้น และแสดงผลจากการคำนวณด้วยเลขทศนิยม 2 ตำแหน่ง เมื่อ

รายได้จากเงินฝาก = เงินต้น * (1+อัตราดอกเบี้ย)ⁿ

```
#include <stdio.h>
void main() {
    int iR;
    float M,R, Income;
    printf("Enter Invested Money(M):");
    scanf("%f",&M);
    printf("Enter %Invested Rate(R):");
    scanf("%d",&iR);
    R=(float)iR/100;
    Income=M*(1+R);
    printf("Income after 1 year: %.2f\n", Income);
}
```



ตัวดำเนินการระดับบิต(bitwise operator)

- โอเปอเรเตอร์ระดับบิต(bitwise operator) คือ โอเปอเรเตอร์ที่นำค่าแต่ละบิตของโอเปอเรนด์ 2 ตัวมากระทำกันหรืออาจเป็นโอเปอเรเตอร์ที่กระทำกับค่าในระดับบิตของโอเปอเรนด์เดียวกันก็ได้

ตัวดำเนินการ	ความหมาย
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
>>	Shift Right
<<	Shift Left

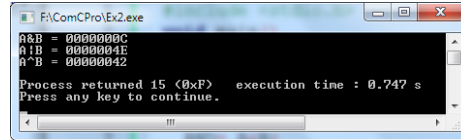
ตัวดำเนินการระดับบิต(bitwise operator)

A	B	A&B	A B	~A	A^B
1	1	1	1	0	0
1	0	0	1	0	1
0	1	0	1	1	1
0	0	0	0	1	0

ตัวดำเนินการระดับบิต(bitwise operator)

- ตัวอย่าง

```
#include <stdio.h>
void main()
{
    unsigned long A, B, AND, OR, XOR;
    A=0X0E;
    B=0X4C;
    AND= A&B;
    printf("A&B = %.8X\n",AND);
    OR= A|B;
    printf("A|B = %.8X\n",OR);
    XOR= A^B;
    printf("A^B = %.8X\n",XOR);
}
```



การดำเนินการแบบ Bit Shift

ตัวดำเนินการ	ความหมาย	ตัวอย่าง
<<	การดำเนินการเลื่อนค่าของตัวแปรไปทางซ้าย	V>>1
>>	การดำเนินการเลื่อนค่าของตัวแปรไปทางขวา	V<<4

```
#include <stdio.h>
void main()
{
    int V=150;
    printf("V=%d\n", V);
    printf("%d x 4= %d\n", V, V<<2);
    printf("%d / 2= %d\n",V, V>>1);
}
```

