บทที่ 3 ตัวดำเนินการ

(Operators and Conversions)

ตัวดำเนินการในภาษา C ส่วนใหญ่จะเหมือนกับตัวดำเนินการในภาษาอื่น สำหรับส่วนแตกต่างที่มีอยู่จะมีทั้งในด้านรูปแบบ และตัวดำเนินการที่ไม่มีในภาษาอื่น ในบทนี้จึงจะกล่าวถึงข้อควรระวังในการใช้ตัวดำเนินการและ ตัวดำเนินการที่เพิ่มในภาษา C เช่น shorthand assignment และ bit manipulation

3.1 ตัวดำเนินการ ตัวถูกดำเนินการ และลำดับของการปฏิบัติการ

ตัวดำเนินการในภาษา C ได้แสดงไว้ในตารางที่ 3.1 จำแนกตามลำดับ สัญลักษณะและหน้าที่ของตัวดำเนินการ ลำดับของตัวดำเนินการที่มีค่าน้อยจะถูกกระทำก่อนลำดับของตัวดำเนินการที่มีค่ามาก กล่าวอีกนัยหนึ่งคือ ตัวดำเนินการที่มีค่าของลำดับน้อยจะมีลำดับความสำคัญสูงกว่า สำหรับในกลุ่มที่มีลำดับความสำคัญเท่ากันหากอยู่ในนิพจน์เดียวกัน จะทำงานจากซ้ายไปขวาและเช่นเดียวกับภาษาอื่นๆ คือหากมีวงเล็บต้องทำงานในวงเล็บก่อน

ตัวถูกดำเนินการของตัวดำเนินการส่วนใหญ่ต้องเป็นชนิดข้อมูลเดียวกันหรือชนิดข้อมูลที่สามารถทำงานร่วมกันได้ แต่หากไม่เป็นไปในลักษณะที่กล่าวถึงนี้ จะมีการแปลงไปสู่ประเภทข้อมูลอื่น ๆ ซึ่งจะถูกกระทำอย่างอัตโนมัติ และตรงไปตรงมา ซึ่งจะกล่าวถึงต่อไป

ลำดับ	สัญลักษณ์ตัวดำเนินการ	หน้าที่
1	++, - -	Postfix/Prefix increment and decrement
1	*(type)*	Cast to type
1	~	Bitwise negation
1	!	Logical NOT
1	-	Unary minus
1	&	Address of
1	*	Indirection (dereferencing)
2	* , / , %	Multiply, divide, modulus
3	+ , -	Addition, subtraction
4	>>, <<	Left, right shift
5	<, >, <=, >=	Test for inequality
6	==, !=	Test for equality, inequality
7	&	Bitwise AND
8	^	Bitwise XOR (exclusive OR)
9	|	Bitwise OR
10	&&	Logical AND
11	||	Logical OR
12	? :	Conditional operator
13	=	Assignment
13	+=, -=, *=, /=, %=	Add, Subtract, multiply, divide, mod
13	>>=, <<=	Shift right and left
13	^=, &=, |=	Bitwise XOR, AND, OR
14	,	The comma operator—sequential evaluation of expressions.

ตารางที่ 3.1 ตัวดำเนินการใน ภาษา C จำแนกตามลำดับการปฏิบัติการและหน้าที่

3.2 ตัวดำเนินการคณิตศาสตร์

ตัวดำเนินการคณิตศาสตร์ ประกอบด้วย

ตัวดำเนินการ	ความหมาย
+	บวก
-	ลบ
*	คูณ
/	หาร
%	ตัวดำเนินการมอดูลัส

การหารแบบจำนวนเต็ม จะให้ผลลัพธ์เป็นจำนวนเต็มโดยปัดเศษจากการหารทิ้ง ขณะที่ตัวดำเนินการมอดูลัส ผลลัพธ์ที่ได้คือเศษที่เหลือจากการหาร ตัวดำเนินการมอดูลัสจะดำเนินการกับข้อมูลประเภท float หรือ double ไม่ได้

เนื่องจาก *, /, % มีลำดับของความสำคัญสูงกว่า + - นั่นแปลว่า หากเขียน

```
salary = salary – salary * tax;
```

นิพจน์ข้างต้นจะทำงาน ดังนี้

```
salary = salary – (salary * tax);
```

3.2.1 การดำเนินการทางคณิตศาสตร์กับจำนวนเต็ม

โดยปกติการปฏิบัติการคณิตศาสตร์ กับจำนวนเต็ม จะได้ผลลัพธ์ที่ถูกต้อง แต่สิ่งที่เราต้องคำนึงคือค่าของจำนวนเต็มที่จะจัดเก็บไม่ควรเกินขนาดที่จะสามารถจัดเก็บได้เพราะถ้าเกิดปัญหาค่าล้นออกมา (overflow value) หากผลลัพธ์จากการคำนวณเป็นค่าที่ล้นออกมา ภาษา C จะไม่แจ้งข่าวสารใด ๆ ออกมาให้ทราบ **แต่สิ่งที่เกิดขึ้นคือผลลัพธ์ที่ผิดจากความต้องการ**

ในเครื่องคอมพิวเตอร์ใด ๆ การบวก 1 เข้ากับจำนวนเต็มบวกที่ใหญ่ที่สุด จะทำให้เกิดการล้นและได้ผลลัพธ์เป็นจำนวนเต็มลบที่มีค่าใหญ่ที่สุด และในทางกลับกัน หากเราลบ 1 ออกจากจำนวนเต็มลบที่มีค่าใหญ่ที่สุด จะทำให้เกิดสภาวะค่าน้อยเกินเก็บ(underflow value) จะได้ผลลัพธ์เป็นจำนวนเต็มบวกที่ใหญ่ที่สุด ดังตัวอย่างที่ 3.1

ตัวอย่างที่ 3.1 โปรแกรมแสดงสภาวะค่าล้นเกินเก็บกับค่าน้อยเกินเก็บ

```
#include <stdio.h>
main()
{
        int x,y;
        x = 32767;
        y =  1;
        printf ("The value of x     = \t%d\n" , x);
        printf ("The value of y     = \t%d\n" , y);
        printf ("The value of x+y   = \t%d\n" , x + y);
        x = -32768;
        y =  1;
        printf ("The value of x     = \t%d\n" , x);
        printf ("The value of y     = \t%d\n" , y);
        printf ("The value of x+y   = \t%d\n" , x - y);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of x      =    32767
The value of y      =    1
The value of x+y    =    -32768
The value of x      =    -32768
The value of y      =    1
The value of x+y    =    32767
```

จะเห็นว่า การปฏิบัติการทั้งสองลักษณะในตัวอย่างที่ 3.1 ทำให้บิตที่เป็นเครื่องหมายเปลี่ยนไป และการคูณจำนวนขนาดใหญ่สองจำนวนเข้าด้วยกัน อาจได้ผลลัพธ์เป็นค่าที่ล้นออกมาในลักษณะนี้ได้เช่นกัน

ทางป้องกันที่ดีที่สุดคือ ระบุประเภทข้อมูลให้พอเหมาะกับขนาดของข้อมูลที่จะบรรจุ เช่นให้ใช้ประเภทข้อมูล long ในกรณีที่จะบวกจำนวนเต็ม int สองจำนวนและคาดว่าผลลัพธ์มากเกินกว่าที่ประเภทข้อมูล int จะรับได้ ดังตัวอย่างที่ 3.2

ตัวอย่างที่ 3.2 โปรแกรมแสดงการกำหนดประเภทข้อมูลที่มีขนาดพอเหมะกับที่ต้องการ

```
#include <stdio.h>
main()
{
        long x,y;
        x = 32767;
        y =  1;

        printf ("The value of x    = \t%ld\n" , x);
        printf ("The value of y    = \t%ld\n" , y);
        printf ("The value of x+y  = \t%ld\n" , x+y);
        x = -32768;
        y =  1;
        printf ("The value of x    = \t%ld\n" , x);
        printf ("The value of y    = \t%ld\n" , y);
        printf ("The value of x+y  = \t%ld\n" , x-y);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of x      =    32767
The value of y      =    1
The value of x+y    =    32768
The value of x      =    -32768
The value of y      =    1
The value of x+y    =    -32769
```

ในการหารจำนวนเต็ม (ทั้งตัวตั้งและตัวหารเป็นจำนวนเต็ม) ผลลัพธ์ที่ได้จะถูกต้องถ้าผลหารลงตัวแต่ถ้าผลหารไม่ลงตัว ผลลัพธ์จะได้เป็นจำนวนเต็มที่มีการตัดเศษทิ้งไป(truncated) หมายความว่าค่าของผลลัพธ์จริงอาจถูกตัดหายไป เช่น 9/2 จะได้ผลลัพธ์เป็น 4 และ 1/3 จะให้ผลลัพธ์เป็น 0

ขณะที่ตัวดำเนินการมอดูลัสจะหาค่าเศษเหลือ ซึ่งผลลัพธ์ที่ได้คือเศษที่เหลือที่เกิดจากการหารจำนวนที่หนึ่งด้วยจำนวนที่สอง เช่น 5%3 ได้ผลลัพธ์เป็น 2 และ 1%3 จะให้ผลลัพธ์เป็น 1

ตัวอย่างที่ 3.3 โปรแกรมแสดงการใช้ตัวดำเนินการหารจำนวนเต็ม และมอดูลัส

```
#include <stdio.h>

int main()
{
        int x,y;

        x = 8;
        y = 9;

        printf ("The value of x     = \t%d\n" , x);
        printf ("The value of y     = \t%d\n" , y);
        printf ("The value of x/y   = \t%d\n" , x/y);
        printf ("The value of x mod y   = \t%d\n" , x%y);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of x      =    8
The value of y      =    9
The value of x/y    =    0
The value of x mod y    =        8
```

ตัวอย่าง 3.4 โปรแกรมแสดงการใช้ตัวดำเนินการบวก ลบ คูณ หาร และมอดูลัส

```
#include <stdio.h>

main()
{
        int x,y;
        x = 38;
        y =  5;
        printf ("The value of x     = \t%d\n" , x);
        printf ("The value of y     = \t%d\n" , y);
        printf ("The value of x + y = \t%d\n" , x+y);
        printf ("The value of x - y = \t%d\n" , x-y);
        printf ("The value of x * y = \t%d\n" , x*y);
        printf ("The value of x / y = \t%d\n" , x/y);
        printf ("The value of x mod y = \t%d\n" , x%y);

        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of x      =    38
The value of y      =    5
The value of x + y  =    43
The value of x - y  =    33
The value of x * y  =    190
The value of x / y  =    7
The value of x mod y =   3
```

3.2.2 การดำเนินการทางคณิตศาสตร์กับจำนวนจริง

การดำเนินการทางคณิตศาสตร์กับจำนวนจริง (floating point arithmetic)ที่ได้ผลลัพธ์โดยประมาณ จะทำการปัดค่าผลลัพธ์ให้มีขนาดผลลัพธ์พอที่จะบรรจุลงในที่เก็บข้อมูล เช่น

สภาวะล้นเกินเก็บ และน้อยเกินเก็บ อาจเกิดขึ้นได้ในการดำเนินการทางคณิตศาสตร์กับจำนวนจริง ตัวอย่างเช่นการบวกจำนวนใด ๆ เข้าไปกับจำนวนจริงที่ใหญ่ที่สุดจะทำให้เกิดสภาวะล้นเกินเก็บ และการหารจำนวนจริง float ที่เล็กที่สุด ด้วยค่าใหญ่ๆ จะทำให้เกิดสภาวะน้อยเกินเก็บ ซึ่งทำให้ผลลัพธ์ต่างจากความเป็นจริง ดังนั้นผู้ใช้จึงควรหลีกเลี่ยงการเกิดเหตุการณ์ดังกล่าว โดยการใช้ประเภทข้อมูลที่เหมาะสม

การหารจำนวนจริงจะต่างจากการหารจำนวนเต็ม คือ ค่าของผลลัพธ์จะไม่มีการตัดเศษของผลลัพธ์ทิ้ง เช่น 1.0/3.0 หรือ 1/3.0 หรือ 1.0/3 ต่างก็ให้ค่าผลลัพธ์ 0.333333

3.3 ตัวดำเนินการสัมพันธ์(relational operator)

เราใช้ตัวดำเนินการสัมพันธ์(relational operator) เพื่อหาค่าความสัมพันธ์หรือมีการเปรียบเทียบระหว่างค่าสองค่า เช่น

ตัวดำเนินการ	ความหมาย
=	เท่ากัน
!=	ไม่เท่ากัน
>	มากกว่า
<	น้อยกว่า
>=	มากกว่า หรือเท่ากับ
<=	น้อยกว่า หรือเท่ากับ

ผลลัพธ์ที่ได้จากการเปรียบเทียบถ้าเป็นจริงจะให้ค่าเป็น 1 หรือถ้าเป็นเท็จ จะให้คาเป็น 0

ตัวอย่างที่ 3.5 โปรแกรมแสดงการใช้ตัวดำเนินการสัมพันธ์

```
#include <stdio.h>

main()
{
        int x,y;
        x = 45;
        y =  54;
        printf ("The value of  x  is \%d\n  ",x );
        printf ("The value of  y  is \%d\n  ",y );
        printf ("The result of  x == y  is \%d\n  ",x == y);
        printf ("The result of   x > y  is \%d\n  " ,x > y);
        printf ("The result of   x < y  is \%d\n  " ,x < y);
        printf ("The result of  x >= y  is \%d\n  " ,x >= y);
        printf ("The result of  x <= y  is \%d\n  " ,x <= y);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value  of  x       is 45
The value  of  y       is 54
The result of  x == y  is 0
The result of  x > y   is 0
The result of  x < y   is 1
The result of  x >= y  is 0
The result of  x <= y  is 1
```

ตัวดำเนินการสัมพันธ์ “ >” , “ < ”, “ >=” และ “<= “ จะมีลำดับของความสำคัญเท่ากัน แต่จะมีลำดับความสำคัญสูงกว่า == และ != ตัวดำเนินการสัมพันธ์ทั้งหมดนี้ มีลำดับความสำคัญสูงกว่า = (เครื่องหมายใช้ในการกำหนดค่า) ดังนั้นในการทำงานเรามักใช้วงเล็บกำกับ เพื่อยืนยันความประสงค์ในการทำงานว่าต้องการให้เปรียบเทียบหลังจากที่มีการกำหนดค่าให้ข้อมูลเรียบร้อยแล้ว เช่น

ควรเขียนว่า

```
while ( (c=getchar())  != EOF )
putchar(c);
```

เพราะหากเราเขียนว่า

```
while ( c=getchar() != EOF )
putchar(c);
```

จะทำงานดังนี้

```
while ( c = ( getchar()  != EOF) )
putchar(c);
```

ซึ่งจะทำงานผิดจากความต้องการ

3.4 ตัวดำเนินการตรรกะ (logical operator)

ตัวดำเนินการตรรกะใช้เพื่อเปรียบเทียบหรือประมวลนิพจน์ตรรกะ ประกอบด้วย

ตัวดำเนินการ	**ความหมาย**
&&	และ
||	หรือ
!	ไม่ใช่

เมื่อนำเงื่อนไข 2 เงื่อนไขมาเปรียบเทียบกันแล้วจะได้ผลการเปรียบเทียบดังตารางต่อไปนี้

ตัวถูกดำเนินการ		**ผลลัพธ์**	
ตัวถูกดำเนินการ 1	**ตัวถูกดำเนินการ 2**	**ตัวดำเนินการ && (และ)**	**ตัวดำเนินการ || (หรือ)**
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

ตารางที่ 3.2 ผลลัพธ์จากการใช้ตัวดำเนินการและ(&&) และ ตัวดำเนินการหรือ (||)

ตัวดำเนินการตรรกะจะใช้เชื่อมนิพจน์เมื่อมีการเปรียบเทียบหลายนิพจน์ เช่นการเปรียบเทียบค่าของข้อมูลสองตัวว่าค่าหนึ่งมากกว่าหรือน้อยกว่าอีกค่าหนึ่ง ในกรณีนี้อาจมีการทดสอบโดยใช้หลาย ๆ นิพจน์ร่วมกันด้วยการใช้ตัวดำเนินการและ(&&) ตัวดำเนินการหรือ (||) และตัวดำเนินการนิเสธ (!) เป็นตัวเชื่อม โดยผลลัพธ์จากการดำเนินการที่ใช้ตัวเชื่อม && || และ ! จะให้ค่าเป็นจริง(1) และให้ค่าเป็นเท็จ(0)

การทำงานของตัวดำเนินการตรรกะ จะทำจากซ้ายไปขวา ดังนั้น หากเราเขียนว่า

```
X < MIN  ||  X> MAX &&  X > 0
```

จะทำงานในลักษณะนี้

(X < MIN) || (X > MAX) && (X < 0)

(1) (2) (3)

การทำงานจะทำการทดสอบค่าในวงเล็บซ้ายมือก่อน และทำการทดสอบค่าในวงเล็บถัดมา จากตัวอย่างข้างต้น การทดสอบเป็นจริงก็ต่อเมื่อ

- ผลการทดสอบของแต่ละวงเล็บเป็นจริงทั้งหมด
- ผลการทดสอบวงเล็บที่ 1 (ซ้ายสุด) เป็นจริง วงเล็บที่ 2 เป็นเท็จ และวงเล็บที่ 3 เป็นจริง
- ผลการทดสอบวงเล็บที่ 1 (ซ้ายสุด) เป็นเท็จ วงเล็บที่ 2 เป็นจริง และวงเล็บที่ 3 เป็นจริง

และการทดสอบเป็นเท็จก็ต่อเมื่อ

- ผลการทดสอบของแต่ละวงเล็บเป็นเท็จทั้งหมด
- ผลการทดสอบวงเล็บที่ 1 (ซ้ายสุด) เป็นจริง วงเล็บที่ 2 เป็นเท็จ และวงเล็บที่ 3 เป็นเท็จ
- ผลการทดสอบวงเล็บที่ 1 (ซ้ายสุด) เป็นเท็จ วงเล็บที่ 2 เป็นจริง และวงเล็บที่ 3 เป็นเท็จ

ในการเชื่อมนิพจน์ด้วย “&&” การทำงานจะทำจากซ้ายไปขวา หากผลการทดสอบของนิพจน์ซ้ายสุดเป็นจริง เช่น MONTH==1 && DAY ==9 การทำงานจะทำการทดสอบ MONTH==1 ก่อนและจะต้องได้ผลลัพธ์เป็นค่าจริงเท่านั้น แล้วจึงจะทดสอบ DAY==9 หากการทดสอบ MONTH==1 เป็นเท็จแล้ว จะหยุดการทดสอบในนิพจน์ลำดับต่อไปและคืนค่าเป็นเท็จเสมอ

ตัวดำเนินการนิเสธจะให้ผลตรงข้ามกับค่าของตัวถูกดำเนินการเสมอ ซึ่งเมื่อนำเงื่อนไข มาเปรียบเทียบแล้วจะได้ผลการเปรียบเทียบดังตารางต่อไปนี้

ตัวดำเนินการ	นิเสธของตัวดำเนินการ
1	0
0	1

ตารางที่ 3.3 ผลลัพธ์จากการใช้ตัวดำเนินการนิเสธ(!)

ตัวอย่างที่ 3.6 โปรแกรมแสดงการใช้ตัวดำเนินการสัมพันธ์ และตัวดำเนินการตรรกะ

```
#include <stdio.h>
main()
{
        int  Score,Min,Max;
        Score= 49;
        Min = 45;
        Max =  54;
        printf ("The value of  Min  is \%d\n  ",Min );
        printf ("The value of  Max  is \%d\n  ",Max );
        printf ("The value of  Score  is \%d\n  ",Score );
        printf ("The result of  Score > Min  is \%d\n  ",Score > Min ); // true
        printf ("The result of  Score < Max  is \%d\n  ",Score < Max ); //true
        printf ("The result of  Score < Min  is \%d\n  ",Score < Min ); // false
        printf ("The result of  Score > Max  is \%d\n  ",Score > Max ); //false
printf ("The result of  Score > Min and Score < Max  is \%d\n  ",Score > Min &&
Score < Max );                  /* true and true */
        printf ("The result of  Score > Min and Score > Max  is \%d\n  ",Score >
Min && Score > Max );           /* true and false */
        printf ("The result of  Score < Min and Score < Max  is \%d\n  ",Score <
Min && Score < Max );           /* false and true */
        printf ("The result of  Score < Min and Score > Max  is \%d\n  ",Score <
Min && Score > Max );           /* false and false */
                printf ("The result of  Score > Min or Score < Max  is \%d\n
",Score > Min || Score < Max ); /* true or true */
        printf ("The result of  Score > Min or Score > Max  is \%d\n  ",Score >
Min || Score > Max );           /* true or false */
        printf ("The result of  Score < Min or Score < Max  is \%d\n  ",Score <
Min || Score < Max );           /* false or true */
printf ("The result of  Score < Min or Score > Max  is \%d\n  ",Score < Min &&
Score > Max );                  /* false or false */
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of  Min  is 45
The value of  Max  is 54
The value of  Score  is 49
The result of  Score > Min  is 1
The result of  Score < Max  is 1
The result of  Score < Min  is 0
The result of  Score > Max  is 0
The result of  Score > Min and Score < Max  is 1
The result of  Score > Min and Score > Max  is 0
The result of  Score < Min and Score < Max  is 0
The result of  Score < Min and Score > Max  is 0
The result of  Score > Min or Score < Max  is 1
The result of  Score > Min or Score > Max  is 1
The result of  Score < Min or Score < Max  is 1
The result of  Score < Min or Score > Max  is 0
```

3.5 ตัวดำเนินการบิตไวส์ (bitwise operators)

ตัวดำเนินการบิตไวส์ ปรากฏดังตารางที่ 3.4

ตัวดำเนินการ	ความหมาย
&	bitwise AND
|	bitwise OR
^	bitwise XOR
<<	left shift
>>	right shift
>>>	zero fill right shift
~	bitwise component
<<=	left shift assignment
>>=	Right shift assignment
>>>=	zero fill right shift assignment
X&=Y	AND assignment
X |=Y	OR assignment
X ^=Y	XOR assignment

ตารางที่ 3.4 ตัวดำเนินการบิตไวส์

หน่วยประมวลผลของเครื่องคอมพิวเตอร์ทั่วไปจะมีคำสั่งทำงานกับตัวดำเนินการบิตไวส์ได้อย่างรวดเร็ว ดังนั้นการทำงานที่ต้องการความเร็วสูง ควรใช้ตัวดำเนินการเหล่านี้เข้าช่วยในการจัดการการคำนวณบางประเภท อีกทั้งยังประหยัดเนื้อที่ในหน่วยความจำ ในที่นี้จะยกตัวอย่างการทำงานเพียงบางตัวเท่านั้น

- **ตัวดำเนินการเลื่อนบิต**

ตัวดำเนินการเลื่อนบิต(bit shift operators) จะมีอยู่ 2 ตัว คือ left shift “<<” ใช้สำหรับเลื่อนบิตไปทางซ้ายมือ และ right shift “ >>” ใช้สำหรับเลื่อนบิตไปทางขวามือ โดยค่าของตัวถูกดำเนินการจะไม่มีการเปลี่ยนแปลง รูปแบบการใช้งานดังนี้

```
d = c << 1
```

หมายความว่า ให้เลื่อนบิตของตัวแปรชื่อ c ไปทางซ้าย 1 ตำแหน่ง และเก็บค่าที่ได้ไว้ในตัวแปร d

ตัวอย่างที่ 3.7 โปรแกรมแสดงการใช้ตัวดำเนินการเลื่อนบิต

```
#include <stdio.h>

main()
{
        int c,d;
        c = 'A';          /* 01000001=41 (Hexadecimal) */
        printf("The character A in Hexadecimal = %x\n", c);
        d = c << 1;       /*10000010=82 (Hexadecimal) */
        printf("After shift 1 bit left = %x\n", d);
        d = c >>1;        /*00100000=20 (Hexadecimal) */
        printf("After shift 1 bit right =%x\n", d);

        c = 'B';          /*01000010=42 (Hexadecimal) */
        printf("The character B in Hexadecimal = %x\n", c);
        d = c << 1;       /*10000100=84 (Hexadecimal) */
        printf("After shift 1 bit left = %x\n", d);
        d = c >>1;        /*00100001=21 (Hexadecimal) */
        printf("After shift 1 bit right =%x\n", d);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The character A in Hexadecimal = 41
After shift 1 bit left = 82
After shift 1 bit right =20
The character B in Hexadecimal = 42
After shift 1 bit left = 84
After shift 1 bit right =21
```

จำนวนบิตที่จะระบุให้เลื่อน ควรเป็นจำนวนเต็มบวก ซึ่งมีค่า น้อยกว่า หรือเท่ากับ จำนวนบิตภายในตัวแปร

- **ตัวดำเนินการบิตไวส์ทางตรรกะ**

ตัวดำเนินการบิตไวส์ทางตรรกะ (bitwise logical operators)ประกอบด้วย บิตไวส์ AND (&) ซึ่งจะให้ค่าผลลัพธ์เป็น 1 ถ้าบิตทั้งสองเป็น 1 นอกนั้นเป็น 0 บิตไวส์ OR (|) ซึ่งจะให้ค่าผลลัพธ์

เป็น 0 ถ้าบิตทั้งสองเป็น 0 นอกนั้นเป็น 1 และ บิตไวส์ XOR (^) ซึ่งจะให้ค่าผลลัพธ์เป็น 1 ถ้าบิตใดบิตหนึ่งเป็น 1 นอกนั้นเป็น 0 ดังสรุปตามตาราง ดังนี้

ตัวถูกดำเนินการ		ผลลัพธ์		
ตัวถูกดำเนินการ 1	ตัวถูกดำเนินการ 2	&	|	^
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

ตารางที่ 3.4 ผลลัพธ์จากการใช้ตัวดำเนินการบิตไวส์ทางตรรกะ

ตัวอย่างที่ 3.8 โปรแกรมแสดงการใช้ตัวดำเนินการบิตไวส์ทางตรรกะ

```
#include <stdio.h>

main()
{
        int c,d;
        c = 'A';          /*01000001=41 (Hexadecimal) */
        d = 'B';          /*01000010=42 (Hexadecimal) */
        printf("The character A in Hexadecimal = %x\n", c);
        printf("The character B in Hexadecimal = %x\n", d);
        printf("A bitwise AND B    = %x\n", c & d);        /* 01000000 */
        printf("A bitwise OR B     = %x\n", c | d);        /* 01000011 */
        printf("A bitwise XOR B  = %x\n", c ^ d); /* 00000011 */
        return 0;
}  /* End main */
```

ผลลัพธ์ที่ได้

```
The character A in Hexadecimal = 41
The character B in Hexadecimal = 42
A bitwise AND B    = 40
A bitwise OR  B      = 43
A bitwise XOR B  = 3
```

ตัวดำเนินการบิตไวส์อีกตัวหนึ่งคือ บิตไวส์นิเสธ (~) ซึ่งจะทำการสลับค่าบิต จาก 0 เป็น 1 และ จาก 1 เป็น 0 เช่น ถ้ากำหนด

X = 00000000 10011001

~X = 11111111 01100110

3.6 ตัวดำเนินการกำหนดค่า

การกำหนดค่าในภาษา C สามารถใช้ตัวดำเนินการกำหนดค่า(assignment operators)ได้มากกว่า 1 ครั้งในประโยคหนึ่งๆ ซึ่งการกำหนดค่ามีความหมายใช้เพื่อกำหนดค่าทางขวามือให้กับตัวแปรทางซ้ายมือ โดยมีรูปแบบดังนี้

```
a=b
```

จะหมายความว่า นำค่าของ b มาเก็บไว้ในตัวแปร a ทั้งนี้ในภาษา C สามารถใช้ตัวดำเนินการมากกว่าหนึ่งครั้งในประโยคหนึ่งๆ

เช่น หากเราต้องการกำหนดค่าศูนย์ (0) เป็นค่าเริ่มต้นให้กับตัวแปรใด ๆ สามารถเขียนได้ดังนี้

```
a = b = c = 0 ;
```

แปลว่าขณะนี้มีการกำหนดค่า a = 0 b = 0 และ c = 0

การทำงานจะทำจากขวาไปซ้าย ดังนั้นจากนิพจน์ข้างต้น การทำงานจะเป็นดังนี้

```
a = (b = (c = 0));
```

ตัวอย่างที่ 3.9 แสดงการใช้ตัวดำเนินการกำหนดค่า

```
#include <stdio.h>
main()
{
        int a,b,c,d;
        a = b = c = d = 5;
        printf("The value of a =%d\n",a);
        printf("The value of b =%d\n",b);
        printf("The value of c =%d\n",c);
        printf("The value of d =%d\n",d);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of a =5
The value of b =5
The value of c =5
The value of d =5
```

3.7 การเขียนตัวดำเนินการกำหนดค่าแบบลัด

การกำหนดค่าให้กับตัวแปรที่อยู่ทางด้านซ้ายของตัวดำเนินการกำหนดค่า (=) ด้วยนิพจน์ทางด้านขวาที่มีตัวแปรทางด้านซ้ายอยู่ด้วย เรามักจะเขียนในรูปแบบดังนี้

ตัวแปรทางด้านซ้าย=นิพจน์ทางด้านขวาซึ่งจะมีตัวแปรทางด้านซ้ายอยู่ด้วย

เช่น val = val/2 ในภาษา C สาวมารถเขียนนิพจน์กำหนดค่าแบบลัดในรูปแบบ

ตัวแปรทางด้านซ้าย ตัวดำเนินการ = นิพจน์ทางด้านขวา(ที่ไม่ต้องใส่ตัวแปรและตัวดำเนินการ)

ตัวดำเนินการแบบลัด(shorthand assignment operator)จะมีลักษณะดังนี้ + =, - =, * =, / =, % =, >>=, <<=, &=, |= และ ^= การเขียนตัวดำเนินการแบบลัดนี้จะทำงานดังนี้

ค่าของตัวแปรทางด้านซ้าย = ค่าของตัวแปรทางด้านซ้ายนำไปดำเนินการกับค่าของนิพจน์ทางด้านขวา

ลำดับ	เขียนแบบปกติ	เขียนแบบลัด
1	val = val/2;	val/=2;
2	val = val>>1;	val>>=1;
3	table[2*i*j] = table[2*i*j]+newval;	table[2*i*j] += newval;

ตารางที่ 3.5 ตัวอย่างการเขียนตัวดำเนินการแบบลัด

จากตารางที่ 3.5 จะเห็นว่าการเขียนแบบลัดจะทำให้การทำงานกระชับรัดกุมขึ้น โดยจากตัวอย่างลำดับที่ 3 ค่าของ subscript ของตัวแปร table จะถูกเรียกใช้เพียงครั้งเดียว

ตัวอย่างที่ 3.10 โปรแกรมแสดงการใช้ตัวดำเนินการกำหนดค่าแบบลัด

```
#include <stdio.h>

main()
{
        int a,b,x,y;
        a = 5;
        b = 1;
        printf("The value of a =%d\n",a);
        x=a++;          /* x=a; a = a +1; */
        printf("The value of x=a++ is %d\n",x);
        printf("The value of a after above execution is %d\n",a);
        y=++a;          /* a = a+1; y = a; */
        printf("The value of y=++a is %d\n",y);
        printf("The value of a after above execution is %d\n",a);

        printf("The value of b =%d\n",b);
        printf("The value of b++ is %d\n",b);
        b = 1;
        printf("Set b = 1, now b is %d\n",b);
        printf("The value of ++b is %d\n",b);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of a =5
The value of x=a++ is 5
The value of a after above execution is 6
The value of y=++a is 7
The value of a after above execution is 7
The value of b =1
The value of b++ is 1
Set b = 1, now b is 1
The value of ++b is 1
```

3.7 ตัวดำเนินการแบบเติมหลัง [postfix] และแบบเติมหน้า [prefix]

ภาษา C อนุญาตให้มีการเขียนตัวดำเนินการแบบลัด อีกสองแบบสำหรับดำเนินการแบบทั่ว ๆ ไป เช่น การเพิ่มค่า และค่าลดค่า โดยระบุ ++ เป็นการเพิ่มค่าทีละ 1 เข้าในตัวถูกดำเนินการ และ - - เป็นการลบค่าทีละ 1 ออกจากตัวถูกดำเนินการ

เราสามารถเขียนตัวดำเนินการลักษณะนี้ในสองรูปแบบ กล่าวคือ

เติมหน้า : นำหน้าตัวดำเนินการ

เติมหลัง : ตามหลังตัวดำเนินการ

ตัวอย่าง กำหนดให้ n = 5

```
a = n++
```

หมายถึง กำหนดให้ n = 5 และเพิ่มค่า 1 ให้ n ดังนั้นหลังจากการทำงาน n=5 และ a=6

แต่ถ้าเขียน

```
a = ++n
```

หมายถึง เพิ่มค่า 1 ให้ n ก่อน และนำค่าดังกล่าวกำหนดให้ a ดังนั้นหลังการทำงานทั้ง a และ n ต่างมีค่าเท่ากับ 6

เรามักเลือกใช้รูปแบบของการเขียนตัวดำเนินการแบบนี้ในการทำงานกับแถวลำดับ เพราะเป็นการเพิ่มประสิทธิภาพในการเขียนโปรแกรม เช่น

```
                        max = a[i++];
หมายถึง                 max = a[i]; i=i+1;
แต่                     max = a[++i]
จะทำงานดังนี้ i = i + 1; max = a[i]
```

3.8 ตัวดำเนินการชนิดอื่น ๆ

ยังมีตัวดำเนินการที่น่าสนใจอีก 2 ชนิด คือ

ตัวดำเนินการคอมมา

ตัวดำเนินการเงื่อนไข

3.8.1 ตัวดำเนินการคอมมา

เราใช้ตัวดำเนินการคอมมาเพื่อเชื่อมนิพจน์หลาย ๆ ตัวเข้าด้วยกัน มีความสามารถทำให้โปรแกรมมีขนาดเล็กลง เช่นการสลับค่าของตัวแปร x และ y โดยปกติเรามักกำหนดตัวแปรชั่วคราว (temp) เพื่อใช้เก็บค่าที่รอการสลับครั้งต่อไป โดยทำดังนี้

```
temp    =    x;
x       =    y;
y       =    temp;
```

แต่เราสามารถเขียนว่า

```
temp = x ,  x=y , y=temp;
```

ตัวอย่าง

```
while ((c=getchar()) ! = EOF)
        putchar(c);
```

อาจเขียนดังนี้

```
while (c=getchar(), c ! = EOF)
        putchar(c);
```

ในส่วนหลัง จะพบว่าแยกการอ่านค่าตัวอักษรออกจากการทดสอบจุดจบของแฟ้มข้อมูล ตัวดำเนินการคอมมา จะทำงานจากซ้ายไปขวา ในกรณีนี้ค่าขวาสุด (ผลลัพธ์ของการทดสอบ C! = EOF) เป็นตัวควบคุมวงวน while แต่อย่างไรก็ตาม ผลลัพธ์ของการเขียนทั้งสองแบบต่างก็ให้ผลเหมือนกัน และเราอาจพบว่า การเขียนการทดสอบลักษณะนี้สลับไปมาในสถานการณ์ต่าง ๆ ให้เลือกเขียนในรูปแบบที่เห็นว่าอ่านแล้วเข้าใจได้ง่ายที่สุด

ในกลุ่มของตัวดำเนินการภาษา C ตัวดำเนินการคอมมามีค่าของลำดับความสำคัญต่ำสุด ดังนั้นในการใช้งานอย่างปลอดภัยเราอาจเลือกใช้วิธีกระจายกลุ่มของนิพจน์ให้เป็นประโยคเดี่ยว (single statement)

หมายเหตุ เครื่องหมายคอมมาที่ใช้แยกตัวพารามิเตอร์แต่ละตัวในฟังก์ชันออกจากกัน ไม่ใช่ตัวดำเนินการคอมมา

ตัวอย่างที่ 3.11 โปรแกรมแสดงการใช้ตัวดำเนินการคอมมา

```
#include <stdio.h>

main()
{
        int  a,b,temp;
        a=25, b =35;
        printf("Before swap\n");
        printf("The value of temp is %d\n", temp);
        printf("The value of a is %d\n",a);
        printf("The value of b is %d\n",b);

        printf("After swap\n");
        temp=a, a=b, b=temp;
        printf("The value of temp is %d\n", temp);
        printf("The value of a is %d\n",a);
        printf("The value of b is %d\n",b);

        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
Before swap
The value of temp is -29180
The value of a is 25
The value of b is 35
After swap
The value of temp is 25
The value of a is 35
The value of b is 25
```

3.8.2 ตัวดำเนินการเงื่อนไข

ตัวดำเนินการเงื่อนไข(conditional operators) ใช้เพื่อทดสอบค่าของนิพจน์ทางตรรกะ โดยมีรูปแบบการใช้ตัวดำเนินการดังนี้

นิพจน์ ? ค่าที่เป็นได้หากนิพจน์มีค่าจริง : ค่าที่เป็นได้หากนิพจน์มีค่าเท็จ

การทำงานจะทำการหาค่านิพจน์ และหากนิพจน์ดังกล่าวไม่ใช่ศูนย์ จะให้ค่าเป็นจริง นอกนั้นเป็นค่าเท็จ และจะเป็นค่าใดค่าหนึ่งเท่านั้นในการทำงานของนิพจน์หนึ่ง ๆ

กล่าวได้ว่าตัวดำเนินการเงื่อนไข เป็นการเขียนแบบลัด สำหรับคำสั่ง if กรณีใช้ในการทดสอบค่าที่มีทางเป็นไปได้ 2 ค่า เช่น

```
if (x<y)
  min = x;
else
  min = y;
```

สามารถเขียนการทดสอบข้างต้น โดยใช้ตัวดำเนินการเงื่อนไข ได้ดังนี้

```
min = (x < y)? x : y ;
```

ตัวอย่างที่ 3.12 โปรแกรมแสดงการใช้ตัวดำเนินการเงื่อนไข

```
#include <stdio.h>
/*ex212.cpp Conditional Opearator*/
int main()
{
        int x;
        puts ("Enter the number 00-99");
        scanf("%d",&x);
        printf("The value of X is %d\n",x);
        (x >=  50)?printf("The value of X higher than 50\n"):printf("The
value of X lower than 50\n");
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
Enter the number 00-99
35
The value of X is 35
The value of X lower than 50
```

3.9 การแปลงประเภทข้อมูล

เราได้กล่าวถึงผลลัพธ์ที่เปลี่ยนไปของการทำงานที่ประเภทข้อมูลของตัวถูกดำเนินการสองตัวที่เหมือนกัน เช่น การทำงานของ long และ double หรือการกำหนดค่า long ให้ short เป็นต้น ผลลัพธ์ที่ได้จากการทำงานดังกล่าวจะมีการแปลงค่าโดยอัตโนมัติ การแปลงค่าโดยอัตโนมัติเกิดขึ้นได้ 2 ลักษณะคือ การกำหนดค่าและการทำงานทางคณิตศาสตร์

3.9.1 การแปลงประเภทข้อมูลอย่างอัตโนมัติ

การแปลงประเภทข้อมูลอย่างอัตโนมัติจะเกิดขึ้นทันทีที่มีการกำหนดค่าค่าหนึ่งให้กับตัวแปรใด ๆ โดยภาษา C จะทำการแปลงค่าที่กำหนดให้ตามประเภทของตัวแปรที่อยู่ทางซ้าย โดยที่ผลลัพธ์อาจมีการตัดส่วนเกินออก เช่นการแปลงค่า float ให้เป็น int

การทำงานโดยอัตโนมัตินี้ให้ความสะดวกแก่ผู้ใช้ แต่ก็สามารถสร้างความยุ่งยากให้เช่นกัน เช่น การกำหนดค่าที่ยาวกว่าให้กับตัวแปรที่มีความสามารถในการเก็บข้อมูลที่สั้นกว่า จะทำให้ผลลัพธ์ผิดจากความเป็นจริง ดังนั้นควรหลีกเลี่ยงการแปลงประเภทข้อมูลจากประเภทข้อมูลที่ใหญ่กว่าไปสู่ประเภทข้อมูลที่เล็กกว่า

ในการแปลงประเภทข้อมูลในประโยคคำสั่ง การกำหนดของภาษา C คือค่าของนิพจน์ทางด้านขวาจะถูกกำหนดให้อยู่ในประเภทของข้อมูลของตัวแปรทางด้านซ้าย หากตัวถูกกระทำทั้งคู่มีประเภทข้อมูลเหมือนกันจะไม่มีการเปลี่ยนแปลงเกิดขึ้น

ตัวอย่างที่ 3.13 โปรแกรมแสดงการแปลงข้อมูลอย่างอัตโนมัติ

```
#include <stdio.h>
main()
{
        char c;
        int i1,i2;
        long l1;
        float f1;
        double d1;
        c = 'a';i1=c;l1=i1;f1=l1;d1=f1;
        printf("The value of c is %d\n",c);
        printf("The value of i1 is %d\n",i1);
        printf("The value of l1 is %ld\n",l1);
        printf("The value of f1 is %f\n",f1);
        printf("The value of d1 is %lf\n",d1);
        i2=d1;
        printf("The value of i2 is %d\n",i2);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of c is 97
The value of i1 is 97
The value of l1 is 97
The value of f1 is 97.000000
The value of d1 is 97.000000
The value of i2 is 97
```

3.9.2 การแปลงประเภทข้อมูลตามที่กำหนด

หากต้องการบังคับให้ข้อมูลใด ๆ มีประเภทข้อมูลตามต้องการ สามารถทำได้โดยการระบุประเภทข้อมูลที่ต้องการไว้หน้าประเภทข้อมูลจริง ตามรูปแบบดังนี้

(type) expression

เช่น ต้องการแปลง float ให้เป็น int อาจเขียนดังนี้

x = (int) 22.5 + 3;

กรณีนี้ 22.5 จะถูกแปลงเป็น int

ตัวอย่างที่ 3.14 โปรแกรมแสดงการแปลงประเภทข้อมูลตามที่กำหนด

```
#include <stdio.h>

main()
{
        float f1,f2,f3;
        f1=17.25;f2=45.26;
        f3=f1+f2;
        printf("The value of f1+f2 is %f\n",f1+f2);
        printf("The value of int(f1+f2) is %d\n",(int)f3);
        printf("The value of int(f1)+int(f2) is %d\n",(int)f1+(int)f2);
        return 0;
} /* End main */
```

ผลลัพธ์ที่ได้

```
The value of f1+f2 is 55.209998
The value of int(f1)+int(f2) is 55
The value of int(f1)+int(f2) is 54
```

แบบฝึกหัด

1. ฝ่ายดำเนินการของร้านทำขนม กำลังคำนวณจำนวนกล่องที่ต้องจัดซื้อมาใช้ในสัปดาห์หน้า โดยมีเงื่อนไขดังนี้
   - การจัดซื้อกล่องครั้งหนึ่ง ๆ ต้องให้พอใช้ได้ 7 วัน
   - ร้านจะผลิตขนม ได้วันละ 22.5 กิโลกรัม
   - ขนม 1 กล่องบรรจุ 225 กรัม

   ให้เขียนโปรแกรมภาษา C เพื่อคำนวณจำนวนกล่องที่ต้องจัดซื้อ

2. จากโจทย์ข้อ 1 ให้รับจำนวนการผลิตขนมต่อวัน(หน่วยเป็นกิโลกรัม) และปริมาณขนมที่บรรจุในแต่ละกล่อง(หน่วยเป็นกรัม) สำหรับเงื่อนไขอื่น ๆ คงเดิม ให้เขียนโปรแกรมภาษา C เพื่อคำนวณจำนวนกล่องที่ต้องจัดซื้อ

   ตัวอย่างรูปแบบการรับข้อมูลนำเข้า :

   Enter the total weight that can produce in each day (Kilogram) : XXX
   Enter the total weight of snack in each box (grams) : XXX

   ตัวอย่างรูปแบบผลลัพธ์ :

   The number of boxes = XXX

3. จงเขียนโปรแกรมที่สามารถพิมพ์จำนวนทศนิยมสองตำแหน่ง 5 จำนวน ในลักษณะดังต่อไปนี้

```
   7.89
  67.89
 567.89
4567.89
34567.89
```

หมายเหตุ สังเกตุตำแหน่งทศนิยมที่ตรงกัน และหาผลรวมของจำนวนทั้งหมด

4. ในการพิมพ์รายงานของร้านค้าแห่งหนึ่งที่ขายสินค้าเพียง3ประเภทเท่านั้น โดยมีรายละเอียดต้นทุนและราคาขายดังนี้

สินค้าชนิดที่	ต้นทุน	ราคาขาย
1	$18.45	$68.45
2	$ 9.3	$49.3
3	$75.26	$85.26

และกำหนดรูปแบบการพิมพ์ผลลัพธ์ ที่มีการคำนวณหากำไรเป็น%ดังนี้

XYZ Department Store

Item1	Cost	Sale	Profit(%)
1	$XXXX.XX	$XXXX.XX	XX.XX%
2	$XXXX.XX	$XXXX.XX	XX.XX%
3	$XXXX.XX	$XXXX.XX	XX.XX%

ตรึงตำแหน่ง$และจุดทศนิยม