

3. 형 변환 및 연산자

♻ 형 변환(type casting) 이란?

변수나 상수의 값을 꺼내올 때 원하는 형태로 변경해 가져오는 것을 형 변환(type casting)이라 한다. 형 변환에는 묵시적 형 변환과(자동 형 변환)과 명시적 형 변환(강제 형 변환)이 있다. 묵시적 형 변환은 프로그래밍의 연산을 위해 컴파일러가 자동으로 형 변환 작업을 하는 것을 말하며 명시적 형 변환은 프로그래머가 형 변환 내용을 지정해 주는 것을 말한다.

♻ 자동 형 변환(묵시적 형 변환) : 컴파일러에 의해 자동으로 이루어지는 형 변환

1. 자동 형 변환이 발생하는 시점 : 서로 자료형(data type)이 다른 data를 연산처리 할 때 발생
2. 자동 형 변환 규칙

- ①자료형(data type)의 크기가 작은 자료형이 크기가 큰 자료형으로 자동 형 변환 된다.
- ②정수와 실수를 연산할 경우 정수가 실수로 자동 형 변환 된다.
- ③int보다 크기가 작은 자료형은 연산 시 int로 자동 형 변환 된다.
- ④signed 과 unsigned를 연산하면 signed가 unsigned으로 자동 형 변환 된다.

3. 자동 형 변환 규칙에 의한 기본 자료형 들의 형 변환 우선순위

우선순위	자료형이름	우선순위	자료형이름
1	char	6	unsigned int
2	unsigned char	7	long
3	short	8	unsigned long
4	unsigned short	9	float
5	int	10	double

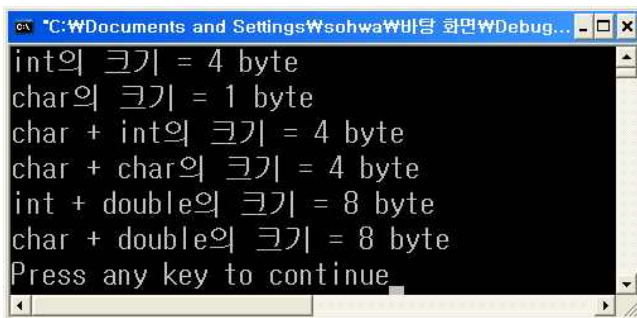
4. 자동 형 변환 관련 예제

예제 3-1 : 자동 형 변환의 결과에 대한 예제

```
#include <stdio.h>
void main ( ) {
    printf("int의 크기 = %d byte\n", sizeof(3) );           // 4가 나온다.
    printf("char의 크기 = %d byte\n", sizeof('A') );       // 1이 나온다.
    printf("char + int의 크기 = %d byte\n", sizeof('A'+1) ); // 4가 나온다.
    printf("char + char의 크기 = %d byte\n", sizeof('A'+'B') ); // 4가 나온다.
    printf("int + double의 크기 = %d byte\n", sizeof(3 + 3.14) ); // 8이 나온다.
    printf("char + double의 크기 = %d byte\n", sizeof('A' + 3.14) ); // 8이 나온다.
}
```

설명 : sizeof() 연산자는 자신의 가로 안에 있는 data의 자료형 크기는 정수로 구해 준다. 즉, sizeof에 int를 넣는다면 (예 : sizeof(15)) 결과로 int의 data type 크기인 4가 나오게 된다. 따라서 위의 예제의 결과는

sizeof(3) → 4가 int임으로 int의 크기인 4가 나온다.
 sizeof('A')); → 'A'가 char임으로 char의 크기인 1이 나온다.
 sizeof('A'+1)); → char + int 임으로 자료형의 크기가 작은 char가 int로 자동 형 변환되어 int+int로 계산 된다. 따라서 결과는 int이며 4가 나온다.
 sizeof('A'+'B')); → char + char 임으로 int보다 작은 자료형은 모두 int로 변경되어 연산되는 자동 형 변환 규칙에 따라 int+int로 계산 된다. 따라서 결과는 int이며 4가 나온다.
 sizeof(3 + 3.14)); → int + double 임으로 자료형의 크기가 작은 int가 double로 자동 형 변환되어 int+int로 계산 된다. 따라서 결과는 int이며 4가 나온다.
 sizeof('A'+3.14)); → char + double 임으로 int보다 작은 자료형은 모두 int로 변경되어 연산되는 자동 형 변환 규칙에 따라 int + double로 계산 된다. 하지만 자료형의 크기가 작은 int가 double로 자동 형 변환되어 int+int로 계산 된다. 따라서 결과는 int이며 4가 나온다.



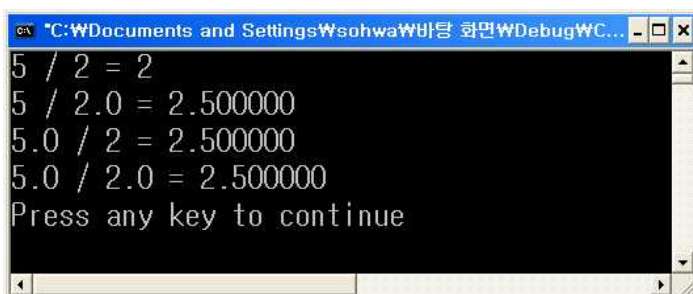
```

int의 크기 = 4 byte
char의 크기 = 1 byte
char + int의 크기 = 4 byte
char + char의 크기 = 4 byte
int + double의 크기 = 8 byte
char + double의 크기 = 8 byte
Press any key to continue
  
```

예제 3-2 : 다른 자료형과 연산 시 발생하는 자동 형 변환

```

#include <stdio.h>
void main ( ) {
    printf("5 / 2 = %d\n", 5 / 2 );           // int/int임으로 결과가 int이다.
    printf("5 / 2.0 = %lf\n", 5 / 2.0 );     // int/double임으로 결과가 double이다.
    printf("5.0 / 2 = %lf\n", 5.0 / 2 );     // double/int임으로 결과가 double이다.
    printf("5.0 / 2.0 = %lf\n", 5.0 / 2.0 ); // double/double임으로 결과가 double이다.
}
  
```

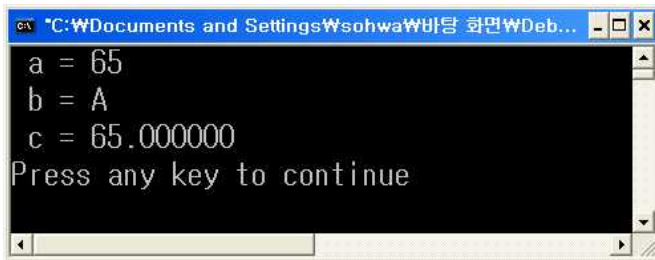


```

5 / 2 = 2
5 / 2.0 = 2.500000
5.0 / 2 = 2.500000
5.0 / 2.0 = 2.500000
Press any key to continue
  
```

예제 3-3 : 다른 자료형에 대입될 경우 발생하는 자동 형 변환

```
#include <stdio.h>
void main ( ) {
    int a;
    char b;
    double c;
    a = 65.78;
    b = 65.78;
    printf(" a = %d \n", a);
    printf(" b = %c \n", b);
    c = b;
    printf(" c = %lf \n", c);
}
```



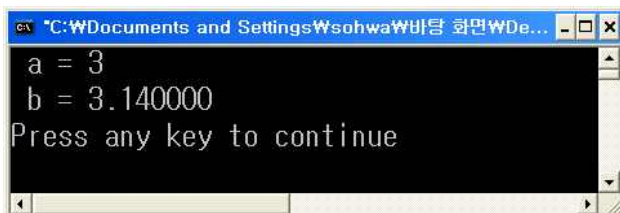
```
*C:\Documents and Settings\sohwa\바탕 화면\Deb...
a = 65
b = A
c = 65.000000
Press any key to continue
```

🔄 강제 형 변환(명시적 형 변환) : 사용자가 직접 형 변환 작업을 하는 것

1. C언어에서의 강제 형 변환 방법 : (자료형)변수명, (자료형)상수명 등.
2. 강제 형 변환 관련 예제

예제 3-4 : 강제 형 변환을 이용한 연산 - 1

```
#include <stdio.h>
void main ( ) {
    int a;
    double b = 3.14 ;
    a = (int) b;
    printf(" a = %d \n", a);
    printf(" b = %lf \n", b);
}
```

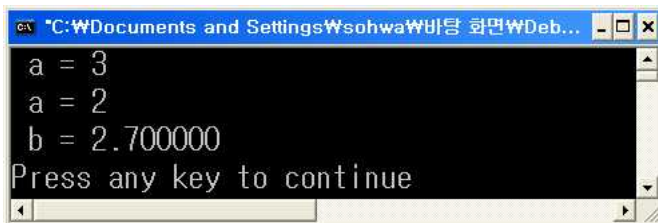


```
*C:\Documents and Settings\sohwa\바탕 화면\De...
a = 3
b = 3.140000
Press any key to continue
```

설명 : 강제 형 변환을 했다 하더라도 원 변수의 값은 변경되지 않는다.

예제 3-5 : 강제 형 변환을 이용한 연산 - 2

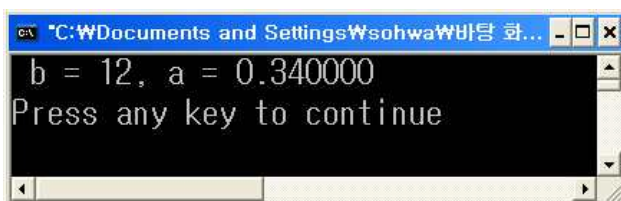
```
#include <stdio.h>
void main ( ) {
    int a;
    double b;
    a = 1.5 + 1.5;
    printf(" a = %d \n", a);
    a = (int)1.5 + (int)1.5;
    printf(" a = %d \n", a);
    b = 1.7 + (int)1.7;
    printf(" b = %lf \n", b);
}
```



```
a = 3
a = 2
b = 2.700000
Press any key to continue
```

예제 3-6 : 강제 형 변환을 이용해 실수를 분리하는 예제

```
#include <stdio.h>
void main ( ) {
    double a = 12.34;
    int b;
    b = (int)a;
    a = a - (double)b;
    printf(" b = %d, a = %lf \n", b, a);
}
```



```
b = 12, a = 0.340000
Press any key to continue
```

🔄 연산자(operator) 란?

자기가 가지고 있는 정의된 기능을 수정한 결과를 가지고 올수 있게 해주는 도구를 말한다.

🔄 C언어의 연산자(operator)

우선순위	연산자의 종류	결합 규칙
1(최우선)	() : 소괄호 [] : 대괄호 . : 도트 (직접멤버 호출 연산자) -> : right arrow (간접멤버 호출 연산자)	왼쪽으로 오른쪽으로 (→)
2	! : not ~ : bit not + : 양수 (단항으로 사용될 경우) - : 부호변경 (단항으로 사용될 경우) ++ : 플러스플러스, 값 1증가 -- : 마이너스마이너스, 값 1감소 & : 레퍼런스 (단항으로 사용될 경우) * : 역참조 (단항으로 사용될 경우) sizeof : 자료를 크기 측정 type cast : 강제 형 변환	오른쪽에서 왼쪽으로 (←)
3	* : 곱하기 / : 나누기 % : 나머지	왼쪽으로 오른쪽으로 (→)
4	+ : 더하기 - : 빼기	왼쪽으로 오른쪽으로 (→)
5	<< : 왼쪽 쉬프트 >> : 오른쪽 쉬프트	왼쪽으로 오른쪽으로 (→)
6	< : 작다 <= : 작거나 같다 > : 크다 >= : 크거나 같다	왼쪽으로 오른쪽으로 (→)
7	== : 같다 != : 같지 않다	왼쪽으로 오른쪽으로 (→)
8	& : bit and (비트논리 곱 연산자)	왼쪽으로 오른쪽으로 (→)
9	^ : bit xor (비트논리 배타적논리합 연산자)	왼쪽으로 오른쪽으로 (→)
10	: bit or (비트논리 합 연산자)	왼쪽으로 오른쪽으로 (→)
11	&& : and (일반논리 곱 연산자)	왼쪽으로 오른쪽으로 (→)
12	: or (일반논리 합 연산자)	왼쪽으로 오른쪽으로 (→)
13	?: : 삼항 조건 연산자	왼쪽으로 오른쪽으로 (→)
14	= : 순수대입 연산자 *=, /=, %=, +=, -=, >>=, <<=, &=, ^=, = : 연산대입 연산자.	오른쪽에서 왼쪽으로 (←)
15(최하위)	, : 그리고 (coma 연산자)	왼쪽으로 오른쪽으로 (→)

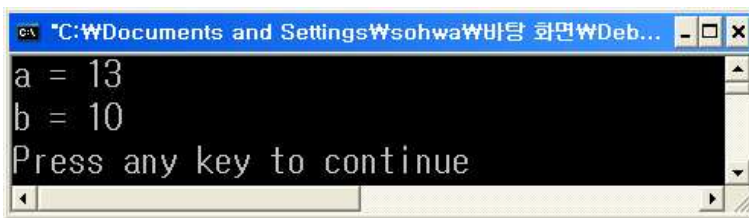
🔄 연산자의 사용방법(operator)

최우선 연산자 : 가장먼저 처리되는 연산자.

1. () 연산자 : 연산의 순서를 지정하거나 조건식을 표기할 때 또는 함수를 호출하거나 함수를 정의할 때 사용한다.

예제 3-7 : () 연산자를 이용한 연산순서 변경

```
#include <stdio.h>
void main ( ) {
    int a, b;
    a = 5 * 3 - 1;           // a에 14가 들어간다.
    b = 5 * (3 - 1);         // b에 10이 들어간다.
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```

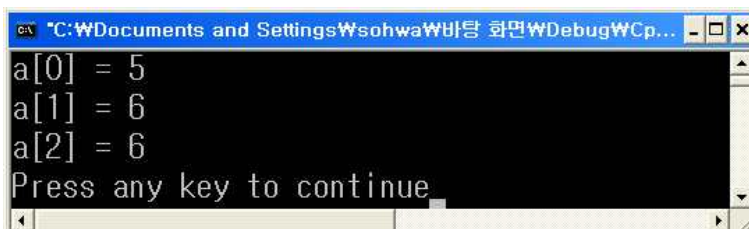


```
C:\Windows and Settings\WsohwaW바탕 화면\WDeb...
a = 13
b = 10
Press any key to continue
```

2. [] 연산자 : 배열을 만들거나 만들어진 배열 내의 원소를 불러오기 위해 사용하는 연산자이다. 배열을 만들 때는 “자료형 배열명[크기]”와 같은 형태로 사용하고 배열을 불러 올 때는 “배열명[인덱스]”와 같은 형태로 원소를 불러온다.

예제 3-8 : [] 연산자의 사용 예

```
#include <stdio.h>
void main ( ) {
    int a[3];               // 크기가 3인 int형 배열을 만든다.
    a[0] = 5;               // 배열의 0번째 원소에 5를 집어 넣는다.
    a[1] = 6;               // 배열의 1번째 원소에 6를 집어 넣는다.
    a[2] = 7;               // 배열의 2번째 원소에 7를 집어 넣는다.
    printf("a[0] = %d\n", a[0]);
    printf("a[1] = %d\n", a[1]);
    printf("a[2] = %d\n", a[1]);
}
```



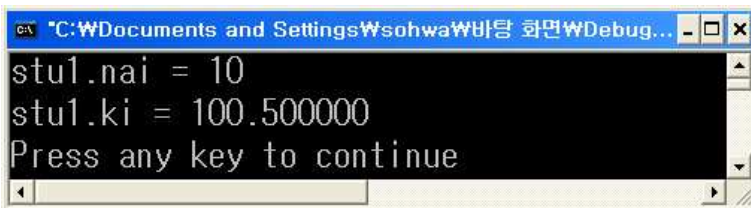
```
C:\Windows and Settings\WsohwaW바탕 화면\WDebugWCp...
a[0] = 5
a[1] = 6
a[2] = 6
Press any key to continue
```

3. . (직접 멤버호출 연산자) 와 -> (간접 멤버호출) 연산자 : . 연산자는 구조체(사용자가 원하는 여러 가지 변수를 한 덩어리로 묶어 하나의 변수처럼 사용할 수 있도록 해주는 C언어의 도구) 형 변수 안에 있는 멤버를 호출하기 위해 사용되며, -> 연산자는 포인터 변수를 이용해 구조체형 변수 안에 있는 멤버를 호출하기 위해 사용한다.

예제 3-9 : . 연산자의 사용 예

```
#include <stdio.h>
struct st {                // 구조체 st를 만든다.
    int nai;               // 첫 번째 멤버인 int nai 선언
    double ki;             // 두 번째 멤버인 double ki 선언
};

void main ( ) {
    struct st stu1;        // struct st형으로 구조체형 변수 stu1을 만든다.
    stu1.nai = 10;         // stu1의 멤버인 nai에 10 을 넣는다.
    stu1.ki = 100.5;       // stu1의 멤버인 ki에 100.5 를 넣는다.
    printf("stu1.nai = %d\n", stu1.nai);
    printf("stu1.ki = %lf\n", stu1.ki);
}
```



예제 3-10 : -> 연산자의 사용 예

```
#include <stdio.h>
struct st {                // 구조체 st를 만든다.
    int nai;               // 첫 번째 멤버인 int nai 선언
    double ki;             // 두 번째 멤버인 double ki 선언
};

void main ( ) {
    struct st stu1;        // struct st형으로 구조체형 변수 stu1을 만든다.
    struct st *sp;         // struct st형 포인터 변수 sp를 만든다.
    sp = &stu1;            // sp에 stu1의 주소값을 넣는다.
    sp->nai = 10;           // sp가 가리키고 있는 구조체형 변수 stu1의 nai에 10을
                           // 넣는다.
    sp->ki = 100.5;         // sp가 가리키고 있는 구조체형 변수 stu1의 ki에 100.5를
                           // 넣는다.
    printf("sp->nai = %d\n", sp->nai);
    printf("sp->ki = %lf\n", sp->ki);
}
```

```

C:\> *C:\Documents and Settings\Wsohwa\바탕 화면\WD...
sp->nai = 10
sp->ki = 100.500000
Press any key to continue

```

단항 연산자 : 하나의 피연산자를 가지는 연산자를 단항 연산자라 한다.

1. ! (not) 연산자 : 일반논리의 NOT 값을 구해 주는 연산자 이다. “!항” 과 같은 식으로 사용하며 항의 값이 참이라면 0을 거짓이라면 1을 결과 값으로 돌려준다. !을 한 번에 여러개 사용하는 것도 가능하다.

※ C언어에서의 자료형별 참, 거짓 표 : C언어에서는 참을 1로 거짓을 0으로 표현한다.

자료형	참(1)인 경우	거짓(0)인 경우
정수 (int, short, long)	0을 제외한 모든 정수 (5, 1, -7, -32767)	0
실수 (float, double)	0을 제외한 모든 값 (100.0, 0.001, -42.15, -0.000001)	0.0 또는 0
문자 (char), 각종 포인터	NULL을 제외한 모든 값	NULL

예제 3-11 : ! 연산자의 사용 예 - 1

```

#include <stdio.h>
void main ( ) {
    int a, b;
    a = !1;           // 참인 1을 ! 했음으로 a에는 0이 들어간다.
    b = !0;           // 거짓인 0을 ! 했음으로 b에는 1이 들어간다.
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}

```

```

C:\> *C:\Documents and Settings\Wsohwa\바탕 화면\WD...
a = 0
b = 1
Press any key to continue

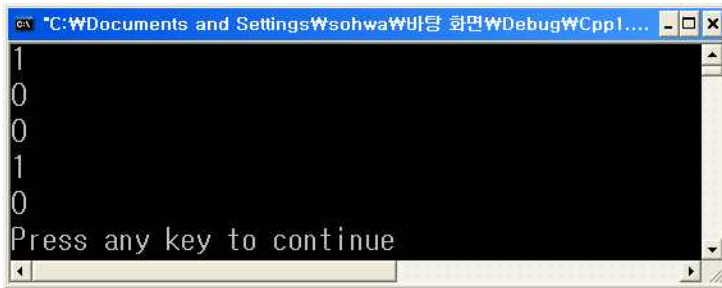
```

예제 3-12 : ! 연산자의 사용 예 - 2

```

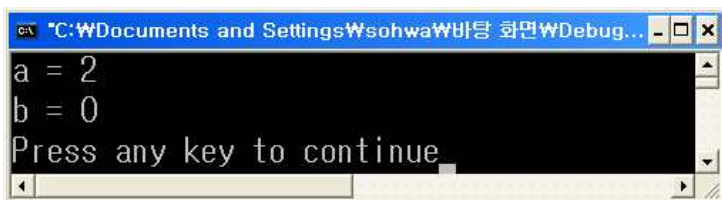
#include <stdio.h>
void main ( ) {
    printf("%d\n", !0);           // 거짓인 0을 ! 했음으로 1
    printf("%d\n", !3);           // 참인 3을 ! 했음으로 0
    printf("%d\n", !'A');         // 참인 'A' 를 !했음으로 0
    printf("%d\n", (3 > 2) );      // 3 > 2는 참임으로 1
    printf("%d\n", !(3 > 2) );     // 참인 3 > 2 를 !했음으로 0
}

```

예제 3-13 : ! 연산자의 사용 예 - 3

```
#include <stdio.h>
void main ( ) {
    int a, b;
    a = !3 + !0 + !!4;      // 3은 참임으로 !하면 0, 0은 거짓임으로 !하면 1,
                           // 4는 참임으로 !하면 0, 0을 다시 !하면 1 이 된다.
                           // 따라서 a에는 0 + 1 + 1 의 결과인 2가 들어간다.
    b = !3 > 6;             // > 보다 !이 우선순위가 높음으로 0 > 6 이 되어 결과가 0
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```



2. ~ (bit not) 연산자 : ~ 연산자는 “~항” 과 같은 식으로 사용하며 항의 비트값을 꺼내와 0을 1로 1을 0으로 하나씩 ! 해주는 연산자 이다. 이때 항은 반드시 정수여야 한다. 예를 들어 변수 a는 short 이며 10이란 값을 가지고 있을 경우 변수 a의 비트 값은 아래와 같다.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

이때 a를 bit not 연산자를 이용해 ~a 연산을 하면 각 비트값 중 0은 1로 변경되고 1은 0으로 바뀌어 아래의 그림과 같은 결과가 나오게 된다.

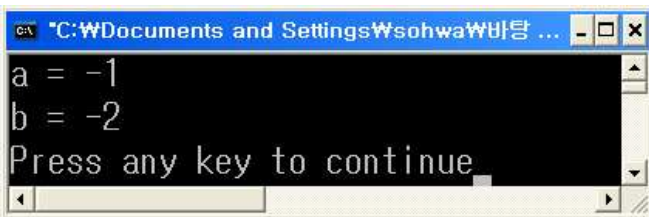
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

위와 같은 비트값을 가지는 정수의 값은 -11 이다.

즉, bit not 연산을 하면 모든 자리의 bit 값이 뒤집어지기 때문에 정수의 부호가 바뀌며 양수는 음수로 변하면서 수의 크기가 1늘어나고, 음수는 양수로 바뀌며 수의 크기가 1감소하게 된다. (예 : ~7 의 결과는 -8이다. ~ -8 의 결과는 7이 된다.)

예제 3-14 : ~ 연산자의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    short a, b;
    a = ~ 0;           // 0을 ~ 하면 -1 이 된다.
    b = ~ 1;           // 1을 ~ 하면 -2 가 된다.
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```



결과설명 :

0을 이진수로 나타내면 아래와 같다. 따라서 ~0의 결과는 그 아래와 같다.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0



2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= -1

1을 이진수로 나타내면 아래와 같다. 따라서 ~1의 결과는 그 아래와 같다.

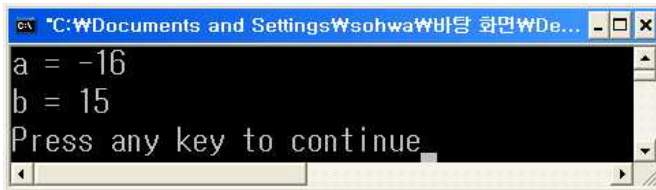
2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	= 1



2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	= -2

예제 3-15 : ~ 연산자의 사용 예 - 2

```
#include <stdio.h>
void main ( ) {
    short a, b;
    a = ~ 15;           // 15를 ~ 하면 -16 이 된다.
    b = ~ -16;          // -16을 ~ 하면 15 가 된다.
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```

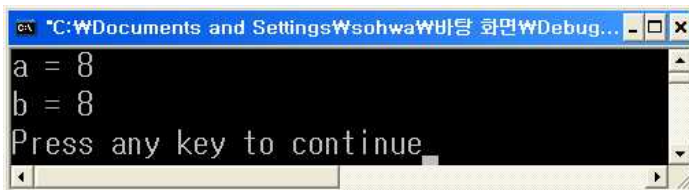


```
a = -16
b = 15
Press any key to continue
```

예제 3-16 : ~ 연산자를 이용한 뺄셈 방법

```
#include <stdio.h>
void main ( ) {
    short a, b;
    a = 15 - 7;
    b = 15 + (~7 +1);          // 빼기 대신 ~한 수를 1 증가시켜 원래 수에 더하면
                                // 뺄셈을 한 것과 같은 결과가 나오게 된다.

    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```

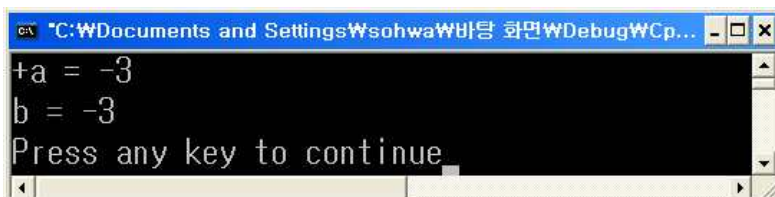


```
a = 8
b = 8
Press any key to continue
```

3. + 연산자 (단항으로 사용될 경우) : 부호를 그대로 유지하게 해준다.

예제 3-17 : + 연산자의 사용 예

```
#include <stdio.h>
void main ( ) {
    short a, b;
    a = -3;
    b = +a;
    printf("+a = %d\n", +a);
    printf("b = %d\n", b);
}
```

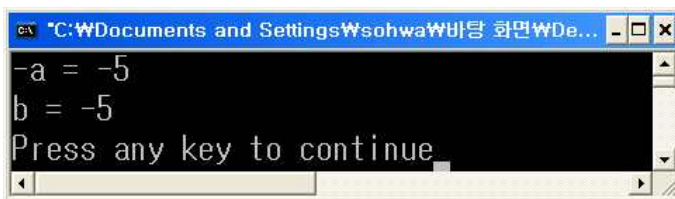


```
+a = -3
b = -3
Press any key to continue
```

4. - 연산자 (단항으로 사용될 경우) : 부호를 바꾼다. 음수는 양수로 양수는 음수로 바뀐다.

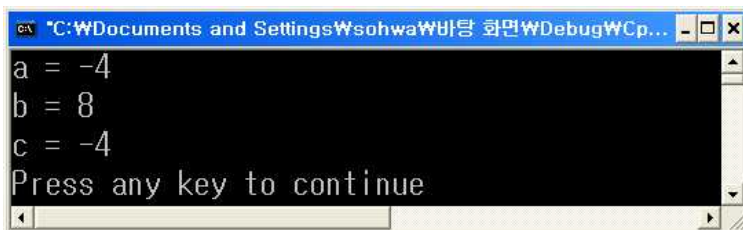
예제 3-18 : - 연산자의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    short a, b;
    a= 5;
    b = -a;
    printf("-a = %d\n", -a);
    printf("b = %d\n", b);
}
```



예제 3-19 : - 연산자의 사용 예 - 2

```
#include <stdio.h>
void main ( ) {
    short a, b, c;
    a= -4;
    b = -a + -a;
    c = - - a;
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %d\n", c);
}
```



5. ++ 연산자, -- 연산자 : 증감연산자라고 불리우며 “++항”, “항++”, “--항”, “항--”와 같은 형태로 사용된다. ++ 연산자는 항의 값을 1 증가시켜 주고 -- 연산자는 항의 값을 1 감소시켜 준다. ++ 과 -- 연산자는 항의 앞쪽에 사용되었을 때와 항의 뒤쪽에 사용되었을 때의 연산 순서가 다르다. 항의 앞쪽에 사용되면 증감연산을 먼저 수행한 뒤 나머지 연산들을 수행하게 되며 항의 뒤쪽에 사용되면 해당 항의 모든 연산이 끝난 뒤 증감연산이 수행되게 된다. 증감 연산자는 바로 값을 증가시킴으로 변수에 대해서만 사용이 가능하며 연산속도가 빠르다.

사용방법	동작설명
++항	전위연산, ++에 의해 항의 값이 1증가 한 뒤 다른 연산이 수행된다.
--항	전위연산, --에 의해 항의 값이 1감소 한 뒤 다른 연산이 수행된다.
항 ++	후위연산, 해당 행의 다른 연산이 모두 수행된 뒤 행이 종료되면서 항의 값이 1증가한다.
항 --	후위연산, 해당 행의 다른 연산이 모두 수행된 뒤 행이 종료되면서 항의 값이 1감소한다.

예제 3-20 : ++과 -- 연산자의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    int a = 5, b = 5;
    a++;                // a의 값이 1증가해 6이 된다.
    b--;                // b의 값이 1감소해 4가 된다.
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```

```
a = 6
b = 4
Press any key to continue
```

예제 3-21 : ++과 -- 연산자의 사용 예 - 2

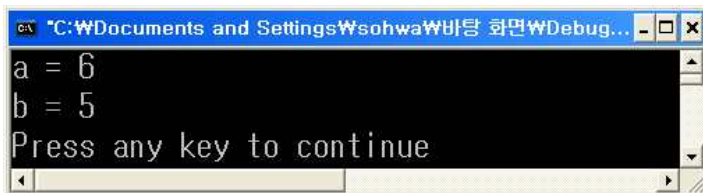
```
#include <stdio.h>
void main ( ) {
    int a = 5, b;
    b = ++a;                // ++이 a의 앞에 있으므로 우선 적으로 a가 1증가해
                           // 6이 된 뒤 그 값이 b에 대입된다. b도 6
    printf("a = %d\n", a);  // 6
    printf("b = %d\n", b);  // 6
}
```

```
a = 6
b = 6
Press any key to continue
```

예제 3-22 : ++과 -- 연산자의 사용 예 - 3

```
#include <stdio.h>
void main ( ) {
    int a = 5, b;
    b = a++;                // ++이 a의 뒤에 있음으로 a의 값이 먼저 b에 대입되어
                           // b가 5가 된 뒤 행이 끝나면서 a가 1증가해 6이 된다.

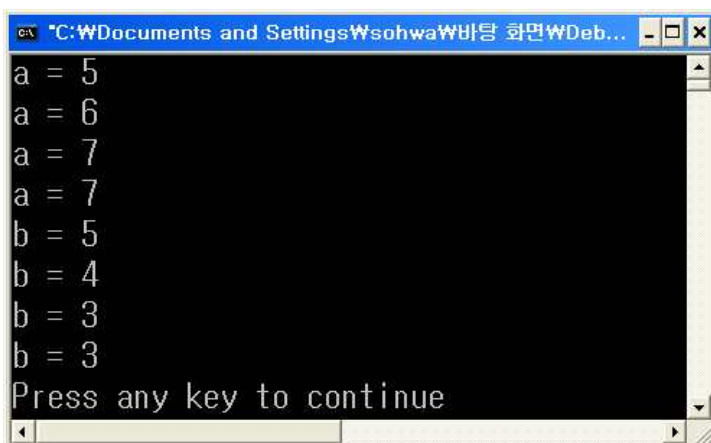
    printf("a = %d\n", a);    // 6
    printf("b = %d\n", b);    // 5
}
```



```
C:\Documents and Settings\sohwa\바탕 화면\Debug...
a = 6
b = 5
Press any key to continue
```

예제 3-23 : 전위 연산과 후위 연산의 차이점

```
#include <stdio.h>
void main ( ) {
    int a = 5, b = 5;
    printf("a = %d\n", a++);    // a를 출력 한 뒤 a가 1증가 됨
    printf("a = %d\n", a);
    printf("a = %d\n", ++a);    // a를 증가 시킨 뒤 출력 함
    printf("a = %d\n", a);
    printf("b = %d\n", b--);    // b를 출력 한 뒤 b가 1감소 됨
    printf("b = %d\n", b);
    printf("b = %d\n", --b);    // b를 감소 시킨 뒤 출력 함
    printf("b = %d\n", b);
}
```

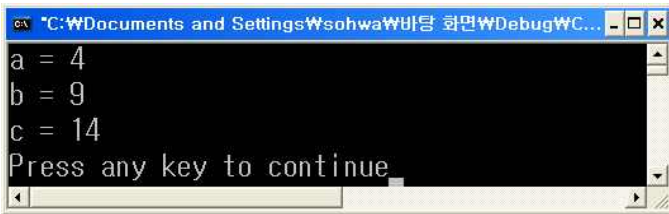


```
C:\Documents and Settings\sohwa\바탕 화면\Debug...
a = 5
a = 6
a = 7
a = 7
b = 5
b = 4
b = 3
b = 3
Press any key to continue
```

예제 3-24 : 증감 연산자의 연산 순서 알아보기 - 1

```
#include <stdio.h>
void main ( ) {
    int a = 5, b = 8, c;
    c = a-- + ++b;          // a는 더해진 후 감소하고 b는 증가한 뒤 더해진다. 따라서
                           // 5 + 9 가 되어 14가 C에 들어 간 이후 a가 4가 된다.

    printf("a = %d\n", a);
    printf("b = %d\n", b);
    printf("c = %d\n", c);
}
```



```
C:\WDocuments and Settings\sohwa\바탕 화면\DebugWC...
a = 4
b = 9
c = 14
Press any key to continue
```

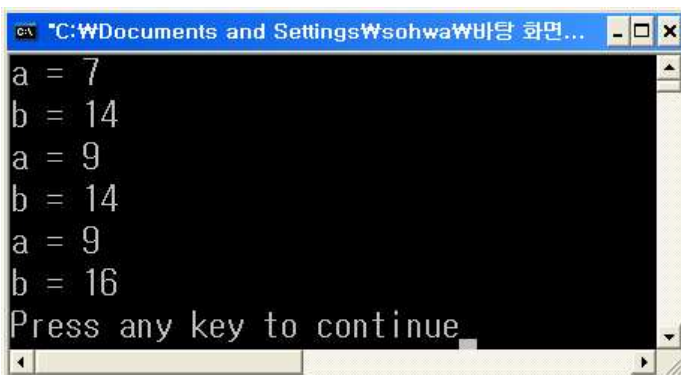
예제 3-25 : 증감 연산자의 연산 순서 알아보기 - 2

```
#include <stdio.h>
void main ( ) {
    int a = 5, b;
    b = ++a + ++a;          // ++ 두 번이 먼저 실행되어 a가 7이 된 뒤 a + a 가 계산됨
                           // 으로 b에는 14가 들어간다.

    printf("a = %d\n", a);  // 7
    printf("b = %d\n", b);  // 14
    b = a++ + a++;          // a + a가 실행 된 뒤 ++이 두 번 연산되어 b에 14가 들어간
                           // 후 a가 증가해 9가 된다.

    printf("a = %d\n", a);  // 9
    printf("b = %d\n", b);  // 14
    b = --a + a++;          // a가 1감소해 8이 된 뒤 a + a 연산을 해 b에 16을 넣고
                           // a가 증가해 9가 된다.

    printf("a = %d\n", a);  // 9
    printf("b = %d\n", b);  // 16
}
```

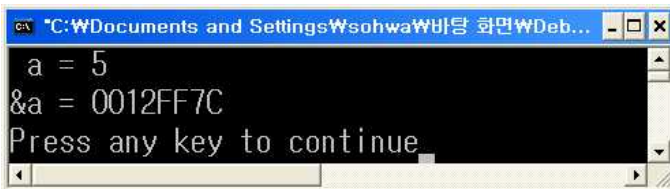


```
C:\WDocuments and Settings\sohwa\바탕 화면...
a = 7
b = 14
a = 9
b = 14
a = 9
b = 16
Press any key to continue
```

7. & (레퍼런스) 연산자 (단항으로 사용될 경우) : 레퍼런스(참조) 연산자라 불리우며 변수의 주소값을 알아보고자 할 경우 사용한다. “&변수명” 과 같은 형태로 사용하며 해당 변수가 보관되어 있는 메모리의 주소값을 돌려준다. 모든 변수의 주소값은 변수의 크기와는 상관없이 변수의 시작주소를 의미한다. C++에서는 별명을 만들기 위해 사용하기도 한다.

예제 3-26 : &연산자의 사용 방법

```
#include <stdio.h>
void main ( ) {
    int a = 5;
    printf(" a = %d\n", a);    // 변수 a에 들어 있는 5가 출력된다.
    printf("&a = %p\n", &a);    // 변수 a의 주소값이 출력된다.
                                // %p는 주소값을 출력하는 printf의 출력서식이다.
}
```



8. * (포인터) 연산자 (단항으로 사용될 경우) : 변수의 주소값을 보관 할 수 있는 포인터 변수를 만들거나 이미 만들어진 포인터 변수가 가지고 있는 주소값을 역참조 하기 위해 사용한다. “자료형 *포인터변수명” 과 같은 형태로 변수를 만들면서 사용하면 해당 자료형을 가지는 변수의 주소값을 보관 할 수 있는 포인터 변수가 만들어 지고 “*포인터변수명” 과 같은 형태로 포인터 변수를 사용하면서 앞쪽에 쓰면 포인터 변수가 가지고 있는 메모리의 주소를 따라 이동해 그 안에 보관되어 있는 data를 꺼내오는 역참조가 된다.

예제 3-26 : *연산자의 사용 방법

```
#include <stdio.h>
void main ( ) {
    int a = 5;
    int *ip;           // int의 주소값을 보관할 수 있는 포인터 변수 ip가 만들어진다.
    ip = &a;           // ip에 변수 a의 주소값을 넣는다. (ip가 a를 가리키게 한다.)
    printf(" a = %d\n", a);    // 변수 a에 들어 있는 5가 출력된다.
    printf(" &a = %p\n", &a);    // 변수 a의 주소값이 출력된다.
    printf(" ip = %p\n", ip);    // ip에 들어 있는 a의 주소값이 출력된다.
    printf("&ip = %p\n",&ip);    // 변수 ip의 주소값이 출력된다.
    printf("*ip = %p\n",*ip);    // ip를 역참조한 값이 출력된다. ip의 역참조 값이란 ip가
                                // 가리키고 있는 변수 a의 값을 말한다.
}
```



```
C:\WDocuments and Settings\sohwa\바탕 화면\Debug...
a = 5
&a = 0012FF7C
ip = 0012FF7C
&ip = 0012FF78
*ip = 5
Press any key to continue
```

산술 연산자 : 피연산자 두 개를 연산하고자 할 때 사용하는 이항 연산자 이다. *, /, %가 +와 -보다 높은 우선순위를 가진다.

1. + (더하기), - (빼기), * (곱하기), / (나누기), % (나머지) 연산자 : 모두 이항으로 사용되었을 경우 산술 연산을 수행한다.

예제 3-27 : +와 - 연산자의 사용 방법

```
#include <stdio.h>
void main ( ) {
    int a = 6, b = 4, c;
    c = a + b;                // 6 더하기 4의 결과인 10 이 c에 들어간다.
    printf(" c = %d\n", c);   // 10
    c = a - b;                // 6 빼기 4의 결과인 2 가 c에 들어간다.
    printf(" c = %d\n", c);   // 2
}
```

```
C:\WDocuments and Settings\sohwa\바탕 화면\Deb...
c = 10
c = 2
Press any key to continue
```

예제 3-28 : *, /, % 연산자의 사용 방법

```
#include <stdio.h>
void main ( ) {
    int a = 6, b;
    b = a * 4;                // 6 곱하기 4의 결과인 24 가 b에 들어간다.
    printf(" b = %d\n", b);   // 24
    b = a / 2;                // 6 나누기 2의 결과인 3 이 b에 들어간다.
    printf(" b = %d\n", b);   // 3
    b = a % 4;                // 6을 4로 나눈 나머지 값인 2가 b에 들어간다.
    printf(" b = %d\n", b);   // 2
}
```

```

C:\WDocuments and Settings\sohwa\바탕 화면\Debug...
b = 24
b = 3
b = 2
Press any key to continue

```

예제 3-29 : / 연산시의 주의 사항 - 1

```

#include <stdio.h>
void main ( ) {
    int a = 5;
    printf("a / 2  = %d\n", a / 2);    // int / int 임으로 결과가 int로 나온다.
    printf("a / 2.0 = %lf\n", a / 2.0); // int / double 임으로 결과가 double로 나온다.
}

```

```

C:\WDocuments and Settings\sohwa\바탕 화면...
a / 2  = 2
a / 2.0 = 2.500000
Press any key to continue

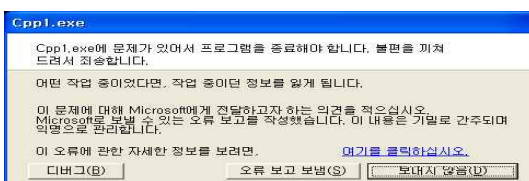
```

예제 3-30 : / 연산시의 주의 사항 - 2

```

#include <stdio.h>
void main ( ) {
    int a = 5;
    printf("a / 0 = %d\n", a / 0); // 0으로 나누면 오류가 발생한다.
}

```



예제 3-31 : 연산시의 우선순위

```

#include <stdio.h>
void main ( ) {
    printf("%d\n", 2 + 3 * 4 + 5 % 2 ); // 2 + 12 + 1 로 계산되어 15가 나온다.
    printf("%d\n", 3 % 4 * 5 );        // 우선순위가 같다면 앞쪽 연산이 먼저 수행된다.
}

```

```

C:\WDocuments and Settings\sohwa\바탕 화면\Debug...
15
15
Press any key to continue

```

쉬프트 연산자

1. << (왼쪽 쉬프트), >> (오른쪽 쉬프트) 연산자 : 정수의 비트 값을 왼쪽 또는 오른쪽으로 원하는 수만큼 쉬프트 하기 위해 사용한다. “변수또는상수명<<정수”, “변수또는상수명>>정수” 와 같은 형태로 사용되며 여기서 사용한 변수 또는 상수는 반드시 정수이어야 한다. 사용 시 변수 또는 상수의 비트값을 뒤쪽에 기술한 정수의 크기만큼 쉬프트 방향으로 이동시킨 값을 구해 준다.

사 용 예	의 미	효 과
A << B	A의 비트값을 왼쪽 방향으로 B만큼 이동시킨다.	쉬프트 1당 * 2
A >> B	A의 비트값을 오른쪽쪽 방향으로 B만큼 이동시킨다.	쉬프트 1당 / 2

시프트 작업 시 비트의 이동으로 인해 발생한 빈 공간에는 0이 채워지며 부호는 쉬프트 되지 않는다.

예 1) 값이 10인 정수 a를 왼쪽으로 두 칸 시프트 했을 경우 : $a \ll 2$

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	= 10



2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	= 40

예 2) 값이 10인 정수 a를 오른쪽으로 두 칸 시프트 했을 경우 : $a \gg 2$

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	= 10



2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	= 2

예 3) 값이 -2인 정수 a를 오른쪽으로 두 칸 시프트 했을 경우 : $a \gg 2$

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	= -2



2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= -1

위의 결과를 보면 오른쪽 쉬프트를 해도 음수는 -1이상 작아지지 않는다는 것을 확인 할 수 있다.

예제 3-32 : << 와 >> 연산자의 사용 방법 - 1

```
#include <stdio.h>
void main ( ) {
    printf(" 4 << 1 = %d\n", 4 << 1);      // 8,   4 * 2와 결과가 같다.
    printf(" 4 << 2 = %d\n", 4 << 2);      // 16,  4 * 4와 결과가 같다.
    printf(" 4 << 3 = %d\n", 4 << 3);      // 32,  4 * 8과 결과가 같다.
    printf(" 4 >> 1 = %d\n", 4 >> 1);      // 2,   4 / 2와 결과가 같다.
    printf(" 4 >> 2 = %d\n", 4 >> 2);      // 1,   4 / 4와 결과가 같다.
    printf(" 4 >> 3 = %d\n", 4 >> 3);      // 0,   4 / 8과 결과가 같다.
}
```

```
C:\Documents and Settings\sohwa\바탕 화면...
4 << 1 = 8
4 << 2 = 16
4 << 3 = 32
4 >> 1 = 2
4 >> 2 = 1
4 >> 3 = 0
Press any key to continue
```

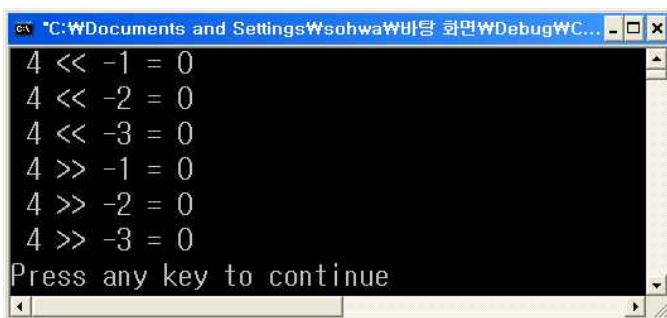
예제 3-33 : << 와 >> 연산자의 사용 방법 - 2

```
#include <stdio.h>
void main ( ) {
    printf(" -4 << 1 = %d\n", -4 << 1);    // -8,   -4 * 2와 결과가 같다.
    printf(" -4 << 2 = %d\n", -4 << 2);    // -16,  -4 * 4와 결과가 같다.
    printf(" -4 << 3 = %d\n", -4 << 3);    // -32,  -4 * 8과 결과가 같다.
    printf(" -4 >> 1 = %d\n", -4 >> 1);    // -2,   -4 / 2와 결과가 같다.
    printf(" -4 >> 2 = %d\n", -4 >> 2);    // -1,   -4 / 4와 결과가 같다.
    printf(" -4 >> 3 = %d\n", -4 >> 3);    // -1에서 더 이상 변하지 않는다.
}
```

```
C:\Documents and Settings\sohwa\바탕 화면\Debug\WCp...
-4 << 1 = -8
-4 << 2 = -16
-4 << 3 = -32
-4 >> 1 = -2
-4 >> 2 = -1
-4 >> 3 = -1
Press any key to continue
```

예제 3-34 : 음수로는 시프트를 할 수 없다.

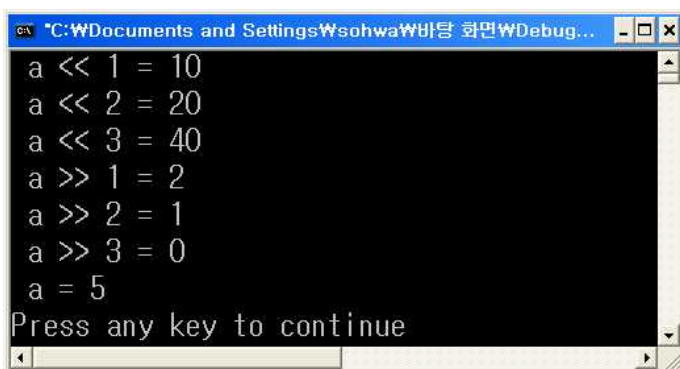
```
#include <stdio.h>
void main ( ) {
    printf(" 4 << -1 = %d\n", 4 << -1);
    printf(" 4 << -2 = %d\n", 4 << -2);
    printf(" 4 << -3 = %d\n", 4 << -3);
    printf(" 4 >> -1 = %d\n", 4 >> -1);
    printf(" 4 >> -2 = %d\n", 4 >> -2);
    printf(" 4 >> -3 = %d\n", 4 >> -3);
}
```



```
C:\Documents and Settings\sohwa\바탕 화면\DebugWC...
4 << -1 = 0
4 << -2 = 0
4 << -3 = 0
4 >> -1 = 0
4 >> -2 = 0
4 >> -3 = 0
Press any key to continue
```

예제 3-34 : 정수형 변수의 시프트 방법

```
#include <stdio.h>
void main ( ) {
    int a = 5;
    printf(" a << 1 = %d\n", a << 1);
    printf(" a << 2 = %d\n", a << 2);
    printf(" a << 3 = %d\n", a << 3);
    printf(" a >> 1 = %d\n", a >> 1);
    printf(" a >> 2 = %d\n", a >> 2);
    printf(" a >> 3 = %d\n", a >> 3);
    printf(" a = %d\n", a );           // a의 값은 그대로 5이다.
}
```



```
C:\Documents and Settings\sohwa\바탕 화면\DebugWC...
a << 1 = 10
a << 2 = 20
a << 3 = 40
a >> 1 = 2
a >> 2 = 1
a >> 3 = 0
a = 5
Press any key to continue
```

관계 연산자 : 이항으로 사용되며 연산자 앞과 뒤쪽 값의 크기를 비교해 0과 1로 결과를 돌려준다.
참이면 1, 거짓이면 0이 된다.

1. < (작다), <= (작거나 같다), > (크다), >= (크거나 같다), == (같다), != (같지않다) 연산자 : 크기를 비교하기 위해 사용하며 “항1 관계연산자 항2”의 형태로 사용한다. 항1 과 항 2를 비교해 사용한 관계연산자 비교가 참이되면 1을 거짓이 되면 0을 결과값으로 돌려준다.

사 용 예	의 미
A < B	A가 B보다 작다면 1(참), 그렇지 않다면 0(거짓)이 나온다.
A <= B	A가 B보다 작거나 같다면 1(참), 그렇지 않다면 0(거짓)이 나온다.
A > B	A가 B보다 크다면 1(참), 그렇지 않다면 0(거짓)이 나온다.
A >= B	A가 B보다 크거나 같다면 1(참), 그렇지 않다면 0(거짓)이 나온다.
A == B	A가 B와 같다면 1(참), 그렇지 않다면 0(거짓)이 나온다.
A != B	A가 B와 같지 않다면 1(참), 그렇지 않다면 0(거짓)이 나온다.

그리고 =< 와 => 는 C언어에 존재하지 않는 연산자임으로 사용 할 수 없다.

예제 3-35 : 관계연산자의 사용 예 - 1

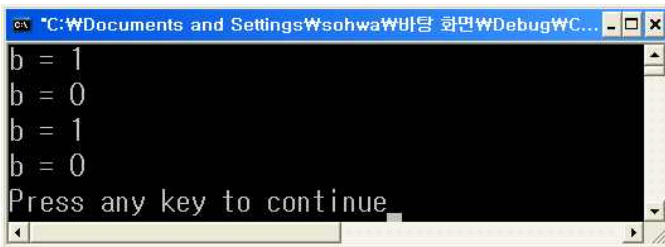
```
#include <stdio.h>
void main ( ) {
    int a;
    printf(" 3 < 5 = %d\n", 3 < 5);
    printf(" 3 < 3 = %d\n", 3 < 3);
    printf(" 3 <= 3 = %d\n", 3 <= 3);
    printf(" 3 <= 2 = %d\n", 3 <= 2);

    printf(" 5 > 3 = %d\n", 5 > 3);
    printf(" 3 > 3 = %d\n", 3 > 3);
    printf(" 3 >= 3 = %d\n", 3 >= 3);
    printf(" 3 >= 5 = %d\n", 3 >= 5);
}
```

```
C:\Documents and Settings\Wsohwa\바탕 화면\WDebug\WCppl.exe
3 < 5 = 1
3 < 3 = 0
3 <= 3 = 1
3 <= 2 = 0
5 > 3 = 1
3 > 3 = 0
3 >= 3 = 1
3 >= 5 = 0
Press any key to continue
```

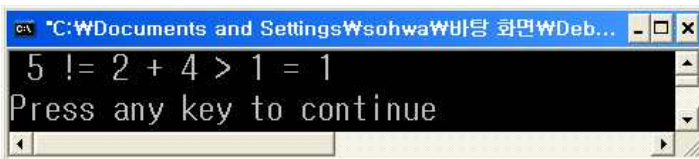
예제 3-36 : 관계연산자의 사용 예 - 2

```
#include <stdio.h>
void main ( ) {
    int a = 3, b;
    b = a > 2;           // 참임으로 b는 1
    printf("b = %d\n", b);
    b = a < 2;           // 거짓임으로 b는 0
    printf("b = %d\n", b);
    b = a == 3;          // 참임으로 b는 1
    printf("b = %d\n", b);
    b = a != 3;          // 거짓임으로 b는 0
    printf("b = %d\n", b);
}
```



예제 3-37 : 관계연산자의 우선순위

```
#include <stdio.h>
void main ( ) {
    printf(" 5 != 2 + 4 > 1 = %d\n", 5 != 2 + 4 > 1 );
    // 산술연산이 우선순위가 높음으로 5 != 6 > 1 로 계산되며 >가 먼저
    // 연산되어 5 != 1이 된다. 따라서 출력은 1
}
```



비트논리 연산자

1. & (bit and), | (bit or), ^ (bit xor) 연산자 : 이항 연산자로 사용되며 각 항의 비트와 비트를 1:1로 대응시켜 연산하는 연산자이다. “항1 비트연산자 항2”와 같은 형태로 사용되며 항1과 항 2는 반드시 정수이어야 한다. 각 연산자의 동작은 다음 표와 같다.

연산자	사용형태	우선순위	결 과
&	A & B	1	양쪽 비트가 모두 1이면 1, 그렇지 않다면 0
^	A ^ B	2	둘 중 하나만 1이면 1, 그렇지 않다면 0
	A B	3	양쪽 비트중 하나라도 1이면 1, 둘다 0이면 0

위의 표 안에 있는 A와 B에 대한 논리연산 결과를 표로 만들면 다음과 같다.

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

위의 비트 연산이 되는 형태를 예로 만들어 보면 다음과 같다.

예 1) A가 10이고 B가 6일 경우 A & B의 연산 모양

$$\begin{array}{l}
 \begin{array}{cccccccccccccccccccc}
 2^{15} & 2^{14} & 2^{13} & 2^{12} & 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 A & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{0} & = 10 \\
 B & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & = 6 \\
 A \& B & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & = 2
 \end{array}
 \end{array}$$

예 2) A가 10이고 B가 6일 경우 A | B의 연산 모양

$$\begin{array}{l}
 \begin{array}{cccccccccccccccccccc}
 2^{15} & 2^{14} & 2^{13} & 2^{12} & 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 A & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{0} & = 10 \\
 B & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & = 6 \\
 A | B & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{1} & \boxed{0} & = 14
 \end{array}
 \end{array}$$

예 3) A가 10이고 B가 6일 경우 A ^ B의 연산 모양

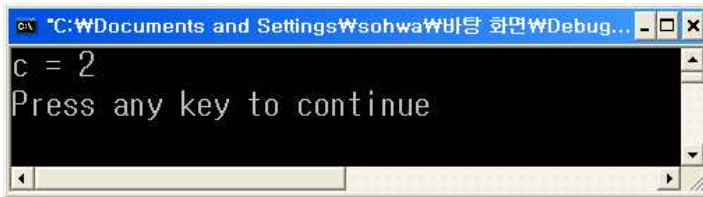
$$\begin{array}{l}
 \begin{array}{cccccccccccccccccccc}
 2^{15} & 2^{14} & 2^{13} & 2^{12} & 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
 A & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} & \boxed{0} & = 10 \\
 B & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & = 6 \\
 A \wedge B & = & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{1} & \boxed{0} & \boxed{0} & = 12
 \end{array}
 \end{array}$$

예제 3-38 : 비트 논리연산자의 사용 예 - 1

```

#include <stdio.h>
void main ( ) {
    int a = 10, b = 6, c;
    c = a & b;
    printf("c = %d\n", c );
}

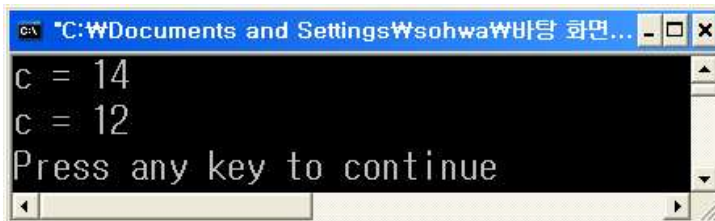
```

```
c = 2
Press any key to continue
```

예제 3-39 : 비트 논리연산자의 사용 예 - 2

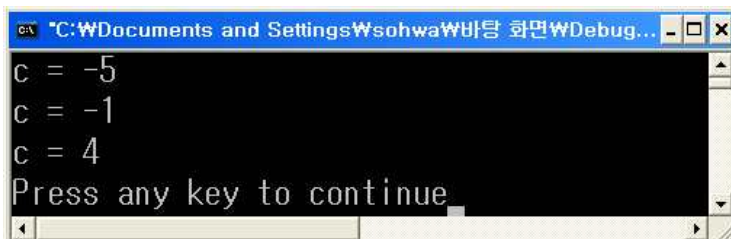
```
#include <stdio.h>
void main ( ) {
    int a = 10, b = 6, c;
    c = a | b;
    printf("c = %d\n", c );
    c = a ^ b;
    printf("c = %d\n", c );
}
```



```
c = 14
c = 12
Press any key to continue
```

예제 3-40 : 비트 논리연산자의 사용 예 - 3

```
#include <stdio.h>
void main ( ) {
    int a = -1, b = -5, c;
    c = a & b;
    printf("c = %d\n", c );
    c = a | b;
    printf("c = %d\n", c );
    c = a ^ b;
    printf("c = %d\n", c );
}
```



```
c = -5
c = -1
c = 4
Press any key to continue
```

	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
A =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= -1
B =	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	= -5
A & B =	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	= -5
A B =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= -1
A ^ B =	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	= 4

일반논리 연산자 : 각 항의 참 또는 거짓 여부를 논리연산 한다.

1. && (and), || (or) 연산자 : 두 개의 항을 필요로 하는 이항연산이며 “항1 일반논리연산자 항2”의 형태로 사용한다. 항1과 항 2의 참 또는 거짓 여부를 판단해 해당 값으로 논리 연산을 한다. && 가 || 보다 우선순위가 높으며 &&의 경우 항1이 거짓이라면 항 2는 실행 되지 않고 바로 0(거짓)이 되며 ||의 경우 항 1이 참이면 항 2는 실행하지 않고 1(참)이 된다.

연산자	사용형태	우선순위	결 과
&&	A && B	1	A와 B가 모두 1이면 1, 그렇지 않다면 0
	A B	2	A와 B중 하나라도 1이면 1, 그렇지 않다면 0

예제 3-41 : 일반 논리연산자의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    printf("0 && 0 = %d\n", 0 && 0 );    // 거짓 && 거짓 이니 결과는 거짓
    printf("0 && 4 = %d\n", 0 && 4 );    // 거짓 && 참 이니 결과는 거짓
    printf("3 && 0 = %d\n", 3 && 0 );    // 참 && 거짓 이니 결과는 거짓
    printf("3 && 4 = %d\n", 3 && 4 );    // 참 && 참 이니 결과는 참
    printf("0 || 0 = %d\n", 0 || 0 );    // 거짓 || 거짓 이니 결과는 거짓
    printf("0 || 4 = %d\n", 0 || 4 );    // 거짓 || 참 이니 결과는 참
    printf("3 || 0 = %d\n", 3 || 0 );    // 참 || 거짓 이니 결과는 참
    printf("3 || 4 = %d\n", 3 || 4 );    // 참 || 참 이니 결과는 참
}
```

예제 3-42 : 일반 논리연산자의 사용 예 - 2

```
#include <stdio.h>
void main ( ) {
    printf("5>2 && 3>2 = %d\n", 5>2 && 3>2 );    // 참 && 참 이니 결과는 참
    printf("5>2 && 3<2 = %d\n", 5>2 && 3<2 );    // 거짓 && 거짓 이니 결과는 거짓
    printf("5>2 || 3>2 = %d\n", 5>2 || 3>2 );    // 참 || 참 이니 결과는 참
    printf("5>2 || 3<2 = %d\n", 5>2 || 3<2 );    // 참 || 거짓 이니 결과는 참
}
```

예제 3-43 : 일반 논리연산자의 사용 예 - 3

```
#include <stdio.h>
void main ( ) {
    int a = 3, b = 3 , c = 3;
    printf("%d\n", a>b && ++c);    // a>b가 거짓임으로 ++c는 실행되지 않는다.
    printf("c = %d\n", c );    // c = 3
}
```

삼항조건 연산자 : 조건에 따라 다른 결과를 얻을 수 있게 해주는 연산자

1. ? : (삼항조건) 연산자 : 3개의 항을 사용하는 연산자 이다. “항1 ? 항2 : 항3”의 형태로 사용하며 항1의 참 또는 거짓 여부를 확인하여 항1이 참이라면 항2가 실행되고 항 1이 거짓이라면 항3이 실행된다. 모든 항들에는 변수나 상수뿐만 아니라 수식이나 함수를 쓸 수도 있다.

예제 3-44 : 삼항조건 연산자의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    int a = 10, b = 5 , c;
    c = b>a ? b : a;    // b가 a보다 크다면 b가 나오고 아니라면 a가 나온다.
    printf("c = %d\n", c );    // c = 10
}
```

```

c = 10
Press any key to continue

```

예제 3-45 : 삼항조건 연산자의 사용 예 - 2

```

#include <stdio.h>
void main ( ) {
    printf("%d\n", 5>2 ? 5*3 : 25/5 ); // 5>2가 참임으로 5*3의 결과인 15가 출력
    printf("%d\n", 2>2 ? 2-5 : 2+5 ); // 2>2가 거짓임으로 2+5의 결과인 7이 출력
}

```

```

15
7
Press any key to continue

```

예제 3-46 : 삼항조건 연산자의 활용 예 -1

```

#include <stdio.h>
void main ( ) {
    int a, b;
    printf("첫번째 정수를 입력해 주세요 : ");
    scanf("%d", &a);
    printf("두번째 정수를 입력해 주세요 : ");
    scanf("%d", &b);
    printf("입력한 두 정수 중 더 큰 수는 %d입니다.\n", a>b ? a : b);
}

```

```

첫번째 정수를 입력해 주세요 : 5
두번째 정수를 입력해 주세요 : 10
입력한 두 정수 중 더 큰 수는 10입니다.
Press any key to continue

```

예제 3-47 : 삼항조건 연산자의 활용 예 - 2

```

#include <stdio.h>
void main ( ) {
    int a;
    printf("하나의 정수를 입력해 주세요 : ");
    scanf("%d", &a);
    a%2==0 ? printf("짝수입니다\n") : printf("홀수입니다\n");
}

```

```

하나의 정수를 입력해 주세요 : 5
홀수입니다
Press any key to continue

```

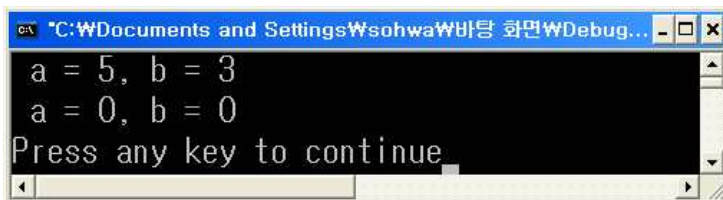
대입 연산자 : 대입연산자는 왼쪽의 변수에 오른쪽 값을 집어넣어 왼쪽 변수의 값을 변경하기 위해 사용하며 **순수 대입**과 **연산 대입**이 있다.

1. **= (순수대입) 연산자** : 순수하게 대입만을 하는 연산자 이다. “항1 = 항2”의 형태로 사용하며 항1에 항2의 결과 값을 집어넣는다. 따라서 항1은 반드시 변수이어야 하며 항2에는 변수, 상수, 수식, 함수 등이 올 수 있다.
2. ***, /=, %=, +=, -=, >>=, <<=, &=, ^=, |= 연산자. (연산대입 연산자)** : 순수대입 연산자와 동작하는 형태는 같지만 자신에게 포함되어 있는 연산자로 연산을 한번 진행 한 뒤 값을 넣는다. 각 연산자별 동작 형태는 다음의 표와 같다.

분 류	연산자	사용예	의 미
순수대입	=	A = B	B의 값을 A에 넣어라
산술대입	+=	A += B	A + B를 연산 한 뒤 그 결과 값을 A에 넣어라
	-=	A -= B	A - B를 연산 한 뒤 그 결과 값을 A에 넣어라
	*=	A *= B	A * B를 연산 한 뒤 그 결과 값을 A에 넣어라
	/=	A /= B	A / B를 연산 한 뒤 그 결과 값을 A에 넣어라
	%=	A %= B	A % B를 연산 한 뒤 그 결과 값을 A에 넣어라
비트논리 대입	&=	A &= B	A & B를 연산 한 뒤 그 결과 값을 A에 넣어라
	^=	A ^= B	A ^ B를 연산 한 뒤 그 결과 값을 A에 넣어라
	=	A = B	A B를 연산 한 뒤 그 결과 값을 A에 넣어라
시프트 대입	<<=	A <<= B	A << B를 연산 한 뒤 그 결과 값을 A에 넣어라
	>>=	A >>= B	A >> B를 연산 한 뒤 그 결과 값을 A에 넣어라

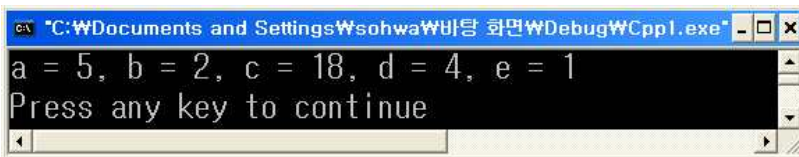
예제 3-48 : 순수 대입 연산자의 사용 예

```
#include <stdio.h>
void main ( ) {
    int a = 5, b = 3;
    printf(" a = %d, b = %d Wn", a, b);
    a = b = 0;
    printf(" a = %d, b = %d Wn", a, b);
}
```



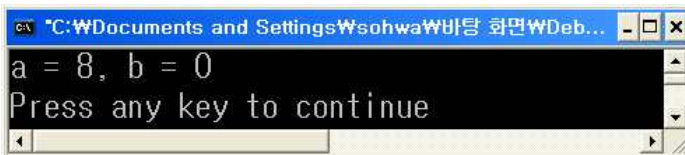
예제 3-49 : 연산 대입 연산자의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    int a = 2, b = 4, c = 6, d = 8, e = 10;
    a += 3;           // a = a + 3 과 같은 의미이다.
    b -= 2;           // b = b - 2 와 같은 의미이다.
    c *= 3;           // c = c * 3 과 같은 의미이다.
    d /= 2;           // d = d / 2 와 같은 의미이다.
    e %= 3;           // e = e % 3 과 같은 의미이다.
    printf("a = %d, b = %d, c = %d, d = %d, e = %d\n", a, b, c, d, e);
}
```



예제 3-50 : 연산 대입 연산자의 사용 예 - 2

```
#include <stdio.h>
void main ( ) {
    int a = 2, b = 2;
    a <<= 2;
    b >>= 2;
    printf("a = %d, b = %d\n", a, b);
}
```



coma 연산자

1. , (그리고) 연산자 : 그리고 라는 뜻을 가지고 있으며 한 행에 여러 동작을 하고 싶거나 특정한 값을 나열하기 위해 사용한다.

예제 3-51 : , 연산자의 사용 예 - 3

```
#include <stdio.h>
void main ( ) {
    int a, b, c;
    a = 5, b = a + 3, c = b % 5;
    printf("a = %d, b = %d, c = %d\n", a, b, c );
}
```

