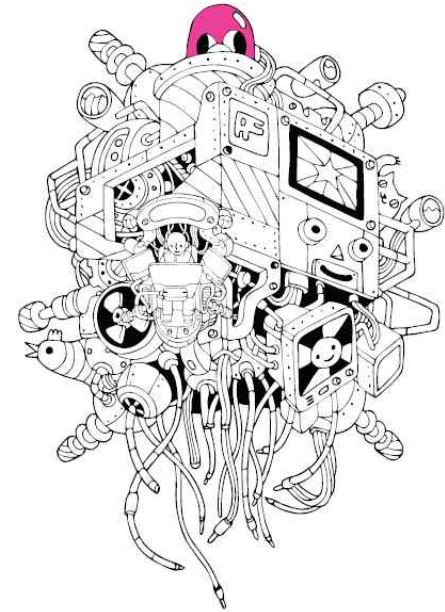


# C 프로그래밍



연산자의 사용

# 기능에 따른 연산자의 분류

| 연산자의 분류    | 연산자             | 의미                       |
|------------|-----------------|--------------------------|
| 대입         | =               | 오른쪽을 왼쪽에 대입              |
| 산술         | + - * / %       | 사칙연산과 나머지 연산             |
| 부호         | + -             |                          |
| 증감         | ++ --           | 증가, 감소 연산                |
| 관계         | > < == != >= <= | 오른쪽과 왼쪽을 비교              |
| 논리         | &&    !         | 논리적인 AND, OR             |
| 조건         | ?               | 조건에 따라 선택                |
| 콤마         | ,               | 피연산자들을 순차적으로 실행          |
| 비트 단위 연산자  | &   ^ ~ << >>   | 비트별 AND, OR, XOR, 이동, 반전 |
| sizeof 연산자 | sizeof          | 자료형이나 변수의 크기를 바이트 단위로 반환 |
| 형변환        | (type)          | 변수나 상수의 자료형을 변환          |
| 포인터 연산자    | * & []          | 주소계산, 포인터가 가리키는 곳의 내용 추출 |
| 구조체 연산자    | . ->            | 구조체의 멤버 참조               |



# 피연산자수에 따른 연산자 분류

---

- ▶ 단항 연산자(unary): 피연산자의 수가 1개

```
++x;  
--y;
```

- 이항 연산자(binary): 피연산자의 수가 2개

```
x + y  
x - y
```

- 삼항 연산자(ternary): 연산자의 수가 3개

```
x ? y : z
```



# 대입(assignment) 연산자와 산술(arithmetic) 연산자

| 연산자 | 연산자의 기능   | 결합방향 |
|-----|---|------|
| =   | 연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다.<br>예) num = 20;                 | ←    |
| +   | 두 피연산자의 값을 더한다.<br>예) num = 4 + 3;                                  | →    |
| -   | 왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다.<br>예) num = 4 - 3;                    | →    |
| *   | 두 피연산자의 값을 곱한다.<br>예) num = 4 * 3;                                  | →    |
| /   | 왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다.<br>예) num = 7 / 3;                   | →    |
| %   | 왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다.<br>예) num = 7 % 3; | →    |

```
int main(void)
{
    int num1=9, num2=2;
    printf("%d+%d=%d \n", num1, num2, num1+num2);
    printf("%d-%d=%d \n", num1, num2, num1-num2);
    printf("%d×%d=%d \n", num1, num2, num1*num2);
    printf("%d÷%d의 몫=%d \n", num1, num2, num1/num2);
    printf("%d÷%d의 나머지=%d \n", num1, num2, num1%num2);
    return 0;
}
```

함수호출 문장에 연산식이 있는 경우  
연산이 이뤄지고 그 결과를 기반으로 함수의 호출이  
진행된다.

9 + 2 = 11  
9 - 2 = 7  
9 × 2 = 18  
9 ÷ 2의 몫 = 4  
9 ÷ 2의 나머지 = 1

실행결과

# 산술(arithmetic) 연산자

- ▶ 덧셈, 뺄셈, 곱셈, 나눗셈 등의 사칙 연산을 수행하는 연산자

| 연산자 | 기호 | 의미               | 예   |
|-----|----|------------------|-----|
| 덧셈  | +  | x와 y를 더한다        | x+y |
| 뺄셈  | -  | x에서 y를 뺀다.       | x-y |
| 곱셈  | *  | x와 y를 곱한다.       | x*y |
| 나눗셈 | /  | x를 y로 나눈다.       | x/y |
| 나머지 | %  | x를 y로 나눌 때의 나머지값 | x%y |



$$y = mx + b$$

$$y = m * x + b$$

$$y = ax^2 + bx + c$$

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

$$m = \frac{x + y + x}{3}$$

$$m = (x + y + z) / 3$$

(참고) 거듭 제곱 연산자는?

C에는 거듭 제곱을 나타내는 연산자는 없다.  
x \* x와 같이 단순히 변수를 두번 곱한다.

# 예제



// 산술 연산자를 이용한 프로그램

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

```
int main(void)
```

```
{
```

```
    int a,b;
```

```
    printf("첫 번째 연산자=");
```

```
    scanf("%d", &a);
```

```
    printf("두 번째 연산자=");
```

```
    scanf("%d", &b);
```

```
    printf("%d + %d 은 %d\n", a, b, a+b);
```

```
    printf("%d - %d 은 %d\n", a, b, a-b);
```

```
    printf("%d * %d 은 %d\n", a, b, a*b);
```

```
    printf("%d / %d 은 %d\n", a, b, a/b);
```

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

```
}
```



첫 번째 연산자=2

두 번째 연산자=3

2 + 3 은 5

2 - 3 은 -1

2 \* 3 은 6

2 / 3 은 0

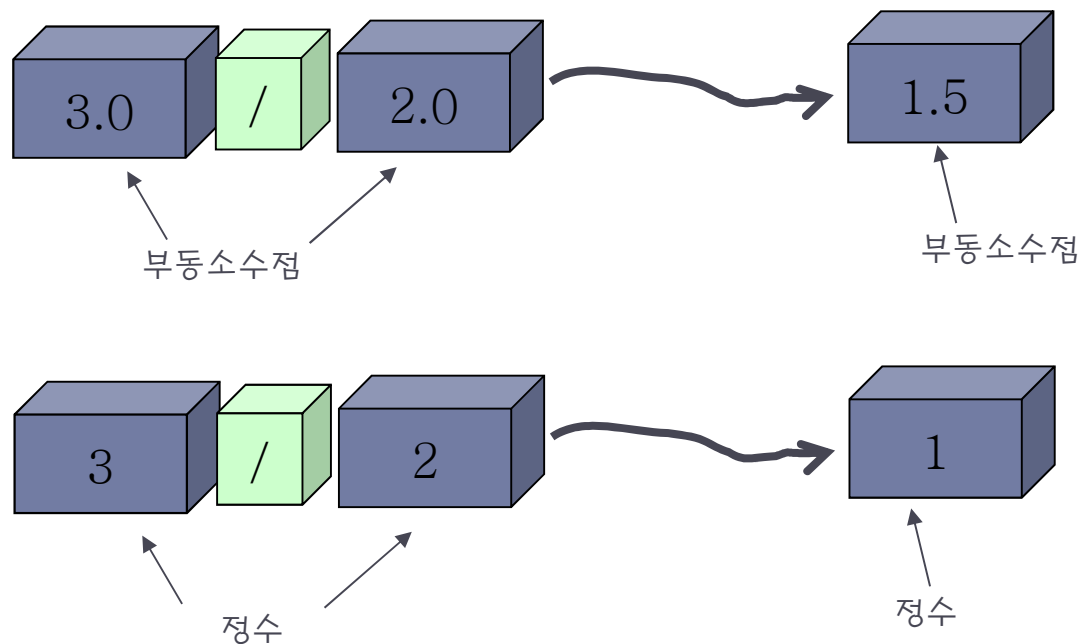
2 % 3 은 2

그렇다면 2, 0 을  
넣었을 때 결과는?

printf("%f / %f = %f\n", a, b, a/b);  
했을 때의 결과는?

나눔셈 연산자

- ▶ 정수형끼리의 나눗셈에서는 결과가 정수형으로 생성하고  
부동소수점형끼리는 부동소수점 값을 생성된다.
- ▶ 정수형끼리의 나눗셈에서는 소수점 이하는 버려진다.



# 나머지 연산자(%)

- ▶ 나머지 연산자(modulus operator)는 첫 번째 피연산자를 두 번째 피연산자로 나누었을 경우의 나머지를 계산
  - ▶  $10 \% 2$ 는 0이다.
  - ▶  $5 \% 7$ 는 5이다.
  - ▶  $30 \% 9$ 는 3이다.
- ▶ 나머지 연산자를 이용한 짝수와 홀수를 구분
  - ▶  $x \% 2$ 가 0이면 짝수
- ▶ 나머지 연산자를 이용한 5의 배수 판단
  - ▶  $x \% 5$ 가 0이면 5의 배수

아주 유용  
한 연산자  
입니다.





# 나눗셈 연산자



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

```
int main()
```

```
{
```

```
    double x, y, result;
```

```
    printf("두개의 실수를 입력하시오: ");
```

```
    scanf("%lf %lf", &x, &y);
```

```
    result = x + y; // 덧셈 연산을 하여서 결과를 result에 대입
```

```
    printf("%f / %f = %f\n", x, y, result);
```

```
    ...
```

```
    result = x / y; // 나눗셈 연산을 하여서 결과를 result에 대입
```

```
    printf("%f / %f = %f\n", x, y, result);
```

```
    printf("%f %% %f = %f\n", x, y, x%y);
```

```
    return 0;
```

```
}
```



두개의 실수를 입력하시오: 7 4

$7.000000 + 4.000000 = 11.000000$

$7.000000 - 4.000000 = 3.000000$

$7.000000 * 4.000000 = 28.000000$

$7.000000 / 4.000000 = 1.750000$

그렇다면 5, 3 을  
넣었을 때 결과는?

`printf("%f / %f = %d\n", x, y, result);`  
했을 때의 결과는?

1. 나머지 연산자 %  
5.000000 % 3.000000 = -1431655765  
5.000000 % 3.000000 = 1.666667

## 나머지 연산자 (%)

```
// 나머지 연산자 프로그램
#include <stdio.h>
#define SEC_PER_MINUTE 60 // 1분은 60초

int main(void)
{
    int input, minute, second;

    printf("초단위의 시간을 입력하시요:(32억초이하) ");
    scanf("%d", &input); // 초단위의 시간을 읽는다.

    minute = input / SEC_PER_MINUTE; // 몇 분
    second = input % SEC_PER_MINUTE; // 몇 초

    printf("%d초는 %d분 %d초입니다. \n", input, minute, second);
    return 0;
}
```



초단위의 시간을 입력하시요:(32억초이하) 70  
70초는 1분 10초입니다.

## 두 피연산자와 연산결과의 자료형은 모두 일치!

- ▶ 두 피연산자의 자료형이 일치해야 하는 이유
  - ▶ 데이터의 표현방식이 일치해야 연산이 가능하도록 CPU 설계
  - ▶ 실수의 데이터 표현 방식과 정수의 데이터 표현 방식이 다르기 때문
- ▶ 피연산자의 자료형과 연산결과의 자료형이 일치하는 이유
  - ▶ 정수 연산의 결과를 실수의 표현방식으로 표현?
  - ▶ 자료형을 근거로 연산 int형 덧셈의 결과는 int, long형 덧셈의 결과는 long

```
int main(void)
{
    int n1= 7;
    double n2 = 1.245;
    double resul = n1+ n2;
    .....
}
```

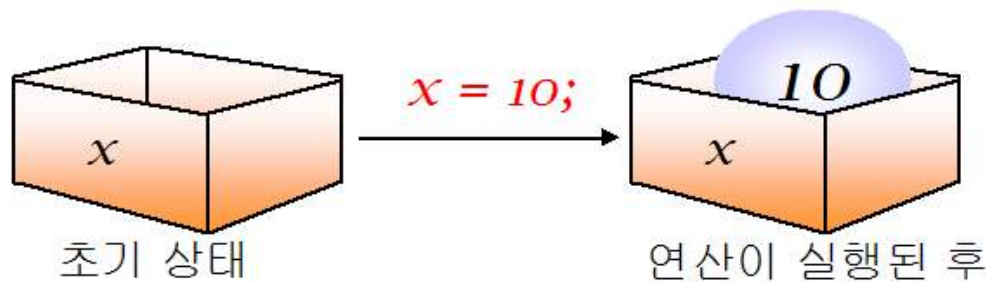
산술 변환!

# 대입(배정, 할당) 연산자

- ▶ 왼쪽에 있는 변수에 오른쪽의 수식의 값을 계산하여 대입

변수(variable) = 수식(expression);

```
x = 10;      // 상수 10을 변수 x에 대입한다.  
y = x;       // 변수 x의 값을 변수 y에 대입한다.  
z = 2 * x + y; // 수식 2 * x + y를 계산하여 변수 z에 대입한다.
```



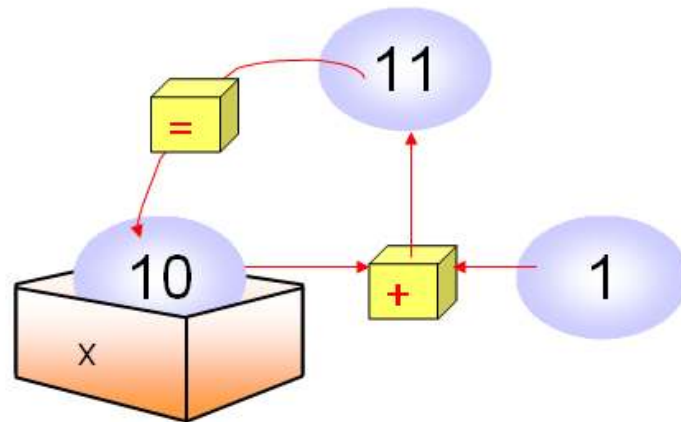
# 대입 연산자 주의점

- ▶ 왼쪽에는 항상 변수가 와야 한다.

$x + 2 = 0;$                       // 왼편이 변수이름이 아니기 때문에 잘못된 수식!!  
 $2 = x;$                          // 왼편이 변수이름이 아니기 때문에 잘못된 수식!!

- 다음의 문장은 수학적으로는 올바르지 않지만 **C**에서는 가능.

$x = x + 1;$                       //  $x$ 의 값이 하나 증가 된다.



# 대입 연산의 결과값

덧셈연산의 결과값은 9

$$x = 2 + 7;$$

대입연산의 결과값은 3(현재는 사용되지 않음)

대입 연산의 결과값은 1

$$y = x = 1;$$

대입 연산의 결과값은 1(현재는 사용되지 않음)

모든 연산에는  
결과값이 있고  
대입 연산도 결  
과값이 있습니다.



# 예제



```
/* 대입 연산자 프로그램 */
```

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

```
int main(void)
```

```
{
```

```
    int x, y;
```

```
    x = 1;
```

```
    printf("수식 x+1의 값은 %d\n", x+1);
```

```
    printf("수식 y=x+1의 값은 %d\n", y=x+1);
```

```
    printf("수식 y=10+(x=2+7)의 값은 %d\n", y=10+(x=2+7) );
```

```
    printf("수식 y=x=3의 값은 %d\n", y=x=3);
```

```
    return 0;
```

```
}
```

수식의 결과값을  
출력하여 보는  
예제입니다.



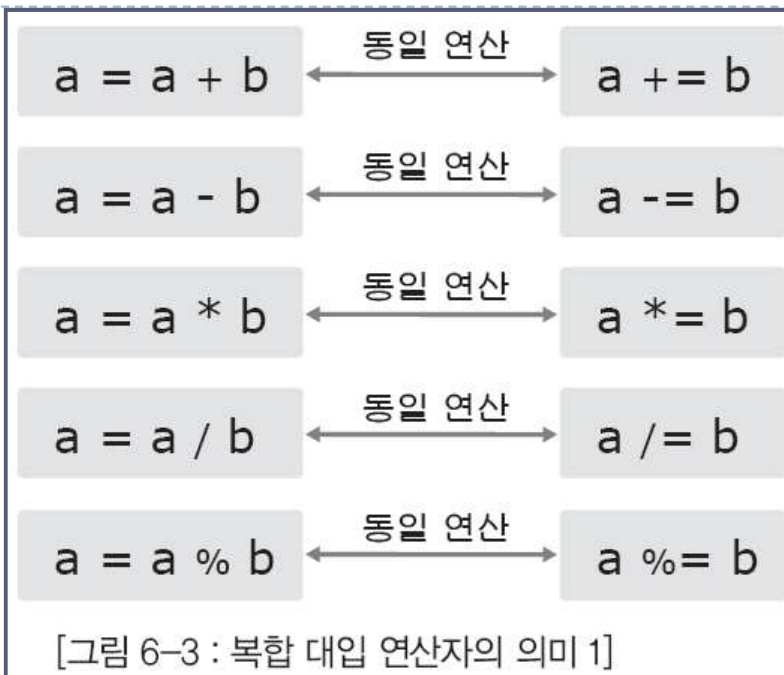
수식 x+ 1의 값은 2

수식 y=x+ 1의 값은 2

수식 y=10+ (x=2+ 7)의 값은 19

수식 y=x=3의 값은 3

## 복합 대입 연산자(+=, -=, \*=, /=, %=)



```
int main(void)
{
    int num1=2, num2=4, num3=6;
    num1 += 3;    // num1 = num1 + 3;
    num2 *= 4;    // num2 = num2 * 4;
    num3 %= 5;    // num3 = num3 % 5;
    printf("Result: %d, %d, %d \n", num1, num2, num3);
    return 0;
}
```

실행결과

Result: 5, 16, 1



# 부호의 의미를 갖는 + 연산자와 - 연산자

```
int main(void)
```

```
{
```

```
    int num1 = +2;
```

```
    int num2 = -4;
```

```
    num1 = -num1;
```

```
    printf("num1: %d \n", num1);
```

```
    num2 = -num2;
```

```
    printf("num2: %d \n", num2);
```

```
    return 0;
```

```
}
```

int num1 = 2; 와 동일한 문장!

+를 연산자의 범주에 포함시켰기 때문에 컴파일이 가능하다.

실행결과

num1: -2

num2: 4

```
num1=-num2;    // 부호 연산자의 사용
```

```
num1-=num2;    // 복합 대입 연산자의 사용
```

두 연산자를 혼동하지 않도록 주의한다.

```
num1 = -num2;    // 부호 연산자의 사용
```

```
num1 -= num2;    // 복합 대입 연산자의 사용
```

혼동을 최소화 하는 띄어쓰기

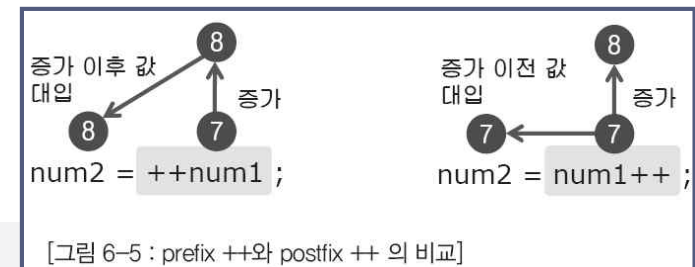
# 증가, 감소 연산자

| 연산자   | 연산자의 기능  | 결합방향 |
|-------|--|------|
| ++num | 값을 1 증가 후, 속한 문장의 나머지를 진행(선 증가, 후 연산)<br>예) val = ++num; | ←    |
| num++ | 속한 문장을 먼저 진행한 후, 값을 1 증가(선 연산, 후 증가)<br>예) val = num++;  | ←    |
| --num | 값을 1 감소 후, 속한 문장의 나머지를 진행(선 감소, 후 연산)<br>예) val = --num; | ←    |
| num-- | 속한 문장을 먼저 진행한 후, 값을 1 감소(선 연산, 후 감소)<br>예) val = num--;  | ←    |

```
int main(void)
{
    int num1=12;
    int num2=12;
    printf("num1: %d \n", num1);
    printf("num1++: %d \n", num1++); // 후위 증가
    printf("num1: %d \n\n", num1);
    printf("num2: %d \n", num2);
    printf("++num2: %d \n", ++num2); // 전위 증가
    printf("num2: %d \n", num2);
    return 0;
}
```

```
num1: 12
num1++: 12
num1: 13

num2: 12
++num2: 13
num2: 13
```



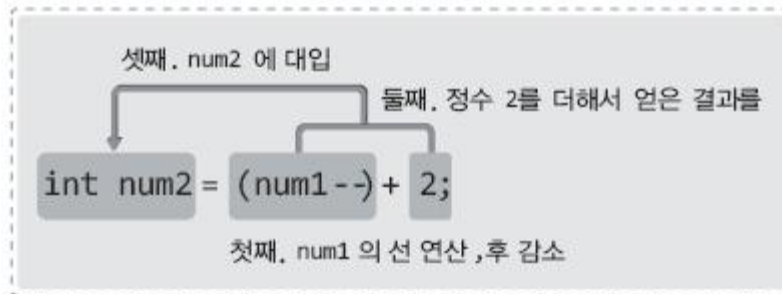
실행결과

## 증가, 감소 연산자 추가 예제

```
int main(void)
{
    int num1=10;
    int num2=(num1--)+2;    // 후위 감소
    printf("num1: %d \n", num1);
    printf("num2: %d \n", num2);
    return 0;
}
```

num1: 9  
num2: 12

실행결과



연산의 과정

```
num1 = 1;
printf("nums : %d, %d, %d", num1++, num1++, num1++);
했을 때의 결과는??
```

## ■ 오~ 이걸 어떻게 해석해야 한단 말인가!

### Bad Case 1

```
int main(void)
{
    int n1, n2;
    n1 = 7;
    n2 = (n1++) + (n1++);
    ....
}
```

해석할 상황을 만들  
지 말자!

### Bad Case 2

```
int main(void)
{
    int num = 1;
    num = num++ ;
    ....
}
```

# 관계 연산자

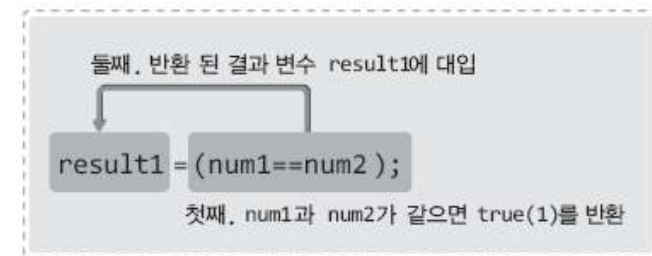
| 연산자 | 연산자의 기능                                  | 결합방향 |
|-----|--|------|
| <   | 예) $n1 < n2$<br>$n1$ 이 $n2$ 보다 작은가?      | →    |
| >   | 예) $n1 > n2$<br>$n1$ 이 $n2$ 보다 큰가?       | →    |
| ==  | 예) $n1 == n2$<br>$n1$ 과 $n2$ 가 같은가?      | →    |
| !=  | 예) $n1 != n2$<br>$n1$ 과 $n2$ 가 다른가?      | →    |
| <=  | 예) $n1 <= n2$<br>$n1$ 이 $n2$ 보다 같거나 작은가? | →    |
| >=  | 예) $n1 >= n2$<br>$n1$ 이 $n2$ 보다 같거나 큰가?  | →    |

C언어는 0이 아닌 모든 값을 참으로 간주한다. 다만 1이 참을 의미하는 대표적인 값일 뿐이다.

실행결과

```
result1: 0
result2: 1
result3: 0
```

연산의 조건을 만족하면 참을 의미하는 1을 반환하고 만족하지 않으면 거짓을 의미하는 0을 반환하는 연산자들이다.



```
int main(void)
{
    int num1=10;
    int num2=12;
    int result1, result2, result3;
    result1=(num1==num2);
    result2=(num1<=num2);
    result3=(num1>num2);
    printf("result1: %d \n", result1);
    printf("result2: %d \n", result2);
    printf("result3: %d \n", result3);
    return 0;
}
```

# 사용예

---

```
1 == 2      // 1과 2가 같으므로 참
1 != 2      // 1와 2가 다르므로 참
1 <= 2      // 1이 2보다 작으므로 참
1 < 2       // 1이 2보다 작으므로 참
(1+2) == (1*2) // (1+2)가 (1*2)와 같지 않으므로 거짓
x >= y      // x가 y보다 크거나 같으면 참
i == 10     // i가 10과 같으면 참
k > 3       // k가 3보다 크면 참
m != 6      // m과 6이 같지 않으면 참
```

False : 0 , True : 1

```
int bool;
bool = (3 == 5);      // bool에는 0이 대입된다.
bool = (3 == 3);      // bool에는 1이 대입된다.
bool = (5 == 5) + (6 != 1); // bool에는 1+1=2가 대입된다.
```



# 예제



```
#include <stdio.h>
int main(void)
{
    int x, y;

    printf("두개의 정수를 입력하시오: ");
    scanf("%d%d", &x, &y);

    printf("x == y의 결과값: %d\n", x == y);
    printf("x != y의 결과값: %d\n", x != y);
    printf("x > y의 결과값: %d\n", x > y);
    printf("x < y의 결과값: %d\n", x < y);
    printf("x >= y의 결과값: %d\n", x >= y);
    printf("x <= y의 결과값: %d\n", x <= y);

    return 0;
}
```



두개의 정수를 입력하시오: 3 4  
x == y의 결과값: 0  
x != y의 결과값: 1  
x > y의 결과값: 0  
x < y의 결과값: 1  
x >= y의 결과값: 0  
x <= y의 결과값: 1

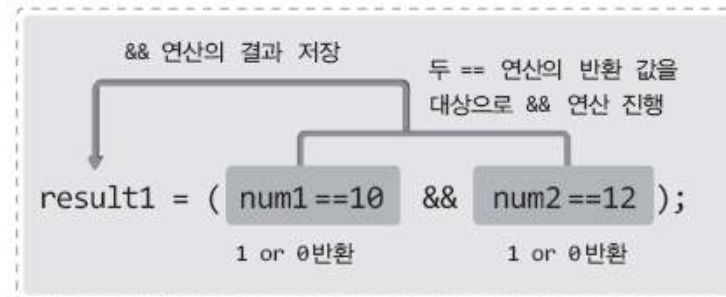
# 논리 연산자

| 연산자 | 연산자의 기능   | 결합방향 |
|-----|---|------|
| &&  | 예) A && B<br>A와 B 모두 '참'이면 연산결과로 '참'을 반환(논리 AND)      | →    |
|     | 예) A    B<br>A와 B 둘 중 하나라도 '참'이면 연산결과로 '참'을 반환(논리 OR) | →    |
| !   | 예) !A<br>A가 '참'이면 '거짓', A가 '거짓'이면 '참'을 반환(논리 NOT)     | ←    |

```
int main(void)
{
    int num1=10;
    int num2=12;
    int result1, result2, result3;
    result1 = (num1==10 && num2==12);
    result2 = (num1<12 || num2>12);
    result3 = (!num1);
    printf("result1: %d \n", result1);
    printf("result2: %d \n", result2);
    printf("result3: %d \n", result3);
    return 0;
}
```

result1: 1  
result2: 1  
result3: 0

실행결과



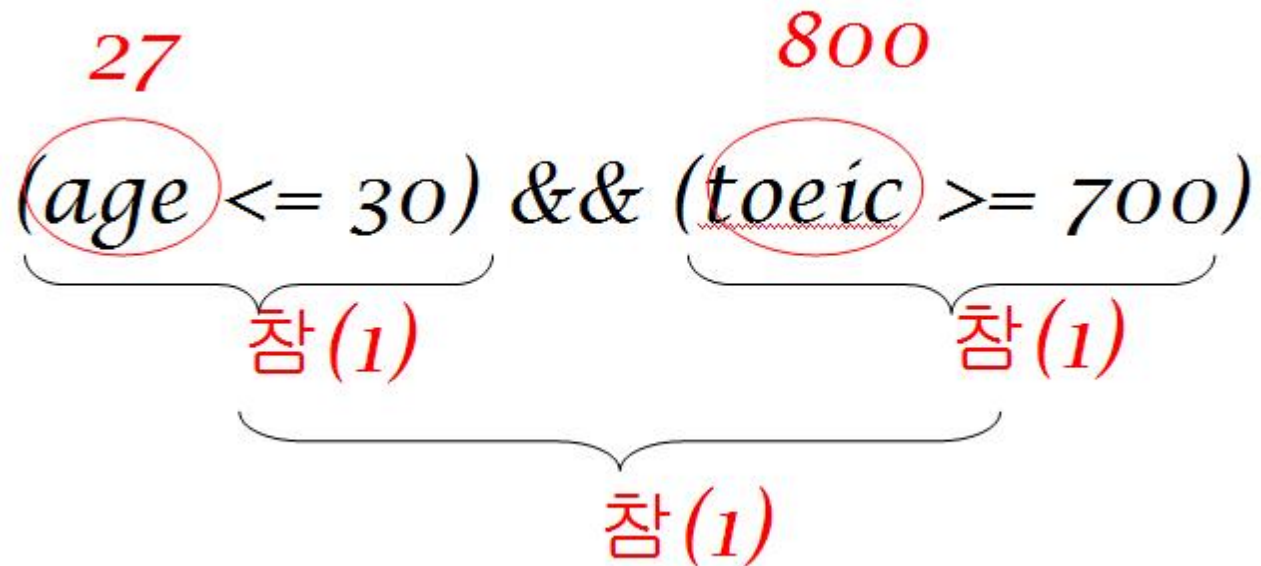
왼쪽 예제에서 num1은 0이 아니므로 참과 거짓의 관계로 본다면 거짓에 해당한다. 따라서 ! 연산의 결과로 참을 의미하는 1이 반환되는 것이다.



## AND 연산자

---

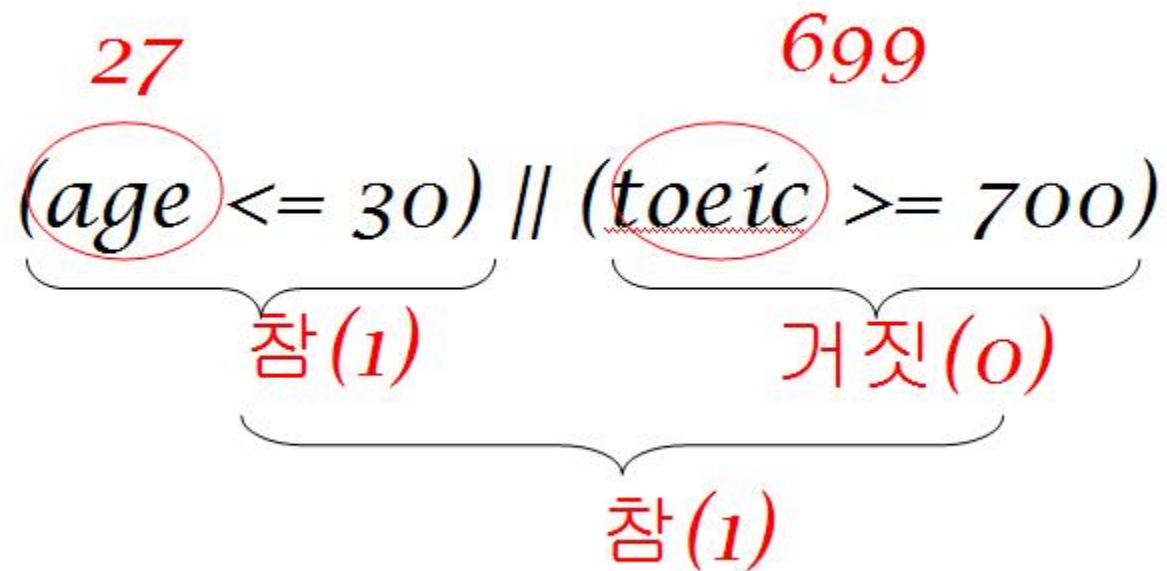
- ▶ 두 개의 피연산자가 모두 참일 때만 연산 결과가 참이 된다



## OR 연산자

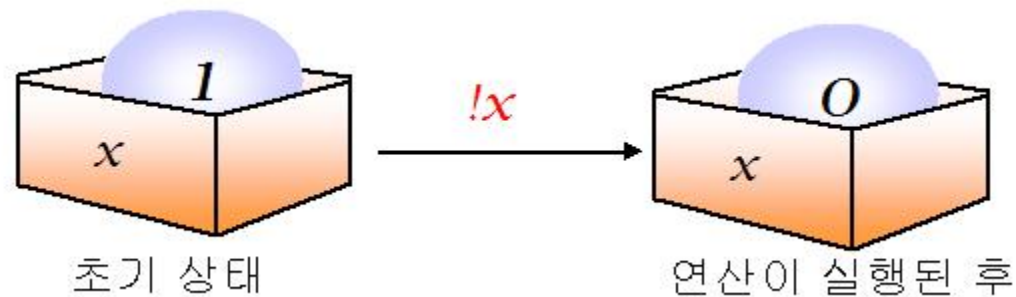
---

- ▶ 하나의 피연산자만 참이면 연산 결과가 참이 된다



# NOT 연산자

- ▶ 피연산자의 값이 참이면 연산의 결과값을 거짓으로 만들고, 피연산자의 값이 거짓이면 연산의 결과값을 참으로 만든다.



- ▶ `result = !1;`      // result에는 0가 대입된다.
- ▶ `result = !(2==3);` // result에는 1이 대입된다.



# coma 연산자

## √ coma( , )

- coma도 연산자이다.
- 둘 이상의 변수를 동시에 선언하거나 둘 이상의 문장을 한 행에 삽입하는 경우에 사용되는 연산자이다.
- 둘 이상의 인자를 함수로 전달할 때 인자의 구분을 목적으로도 사용된다.
- coma 연산자는 다른 연산자들과 달리 연산의 결과가 아닌 '구분'을 목적으로 한다.

```
int main(void)
{
    int num1=1, num2=2;
    printf("Hello "), printf("world! \n");
    num1++, num2++;
    printf("%d ", num1), printf("%d ", num2), printf("\n");
    return 0;
}
```

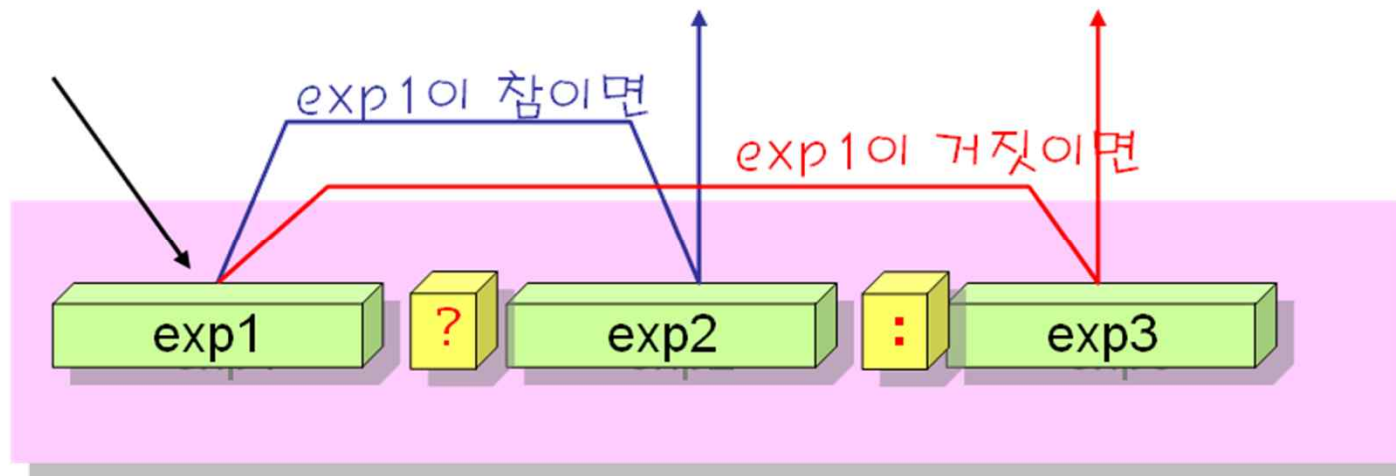
실행결과

```
Hello world!
2 3
```

## 조건 연산자( ? : )

- ▶ exp1가 참이면 exp2를 반환, 그렇지 않으면 exp3를 반환

exp1 ? exp2 : exp3



```
absolute_value = (x > 0) x: -x; // 절대값 계산  
max_value = (x > y) x: y; // 최대값 계산  
min_value = (x < y) x: y; // 최소값 계산
```

# 예제



```
#include <stdio.h>
int main(void)
{
    int x,y;

    printf("첫 번째 수=");
    scanf("%d", &x);
    printf("두 번째 수=");
    scanf("%d", &y);

    printf("큰 수=%d \n", (x > y) ? x : y);
    printf("작은 수=%d \n", (x < y) ? x : y);
}
```

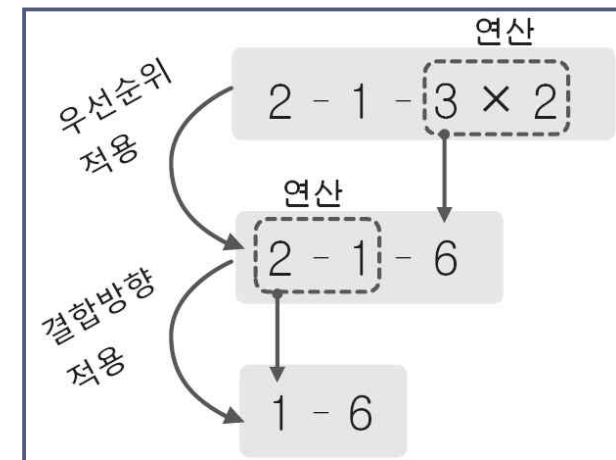


첫 번째 수=2  
두 번째 수=3  
큰 수=3  
작은 수=2

## 연산자의 우선순위 : 우선순위와 결합방향의 이해

- ▶ 연산의 순서를 결정하는 두 가지 요소
  - ▶ 우선순위: 연산자들 사이의 연산순서 결정요소
  - ▶ 결합방향: 우선순위가 같은 연산자들 사이의 연산순서 결정요소
- ▶ 우선순위 같고, 결합방향 다르다?
  - ▶ 우선순위 같으면 결합방향도 같다.
  - ▶ 표 6-2 참조

|    |              |               |   |
|----|--------------|---------------|---|
| 3위 | (casting)    | 자료형 변환        | ← |
| 4위 | *, /, %      | 곱셈, 나눗셈 관련 연산 | → |
| 5위 | +, -         | 덧셈, 뺄셈        | → |
| 6위 | <<, >>       | 비트 이동         | → |
| 7위 | <, >, <=, >= | 대소 비교         | → |
| 8위 | ==, !=       | 동등 비교         | → |



## 연산자의 우선순위 : 순서를 조절하는 구분자

---

### ▶ 소괄호

- ▶ 연산자가 아닌 구분자(separator)이다.
- ▶ 연산의 대상을 별도의 단위로 구분시키는 구분자.
- ▶ 때문에 소괄호 내에 존재하는 연산문에서 하나의 연산 결과를 구성

```
int main(void)
{
    int num = 5 - ( 3 + 2 );
    .....
}
```

3 + 2가  
구분이 됨!





## 예제

---

$$y = \underbrace{a \% b}_{\textcircled{2}} / c + d * \underbrace{(e - f)}_{\textcircled{1}};$$

The diagram illustrates the evaluation order of the expression  $y = a \% b / c + d * (e - f);$  using red lines and numbered circles (1-6) to show the sequence of operations:

- ①:  $(e - f)$
- ②:  $a \% b$
- ③:  $(a \% b) / c$
- ④:  $d * ((a \% b) / c)$
- ⑤:  $(e - f) + (d * ((a \% b) / c))$
- ⑥:  $y = ((e - f) + (d * ((a \% b) / c)))$



# 예제



```
#include <stdio.h>
int main(void)
{

    int x=0, y=0;
    int result;

    result = 2 > 3 || 6 > 7;
    printf("%d\n", result);

    result = 2 || 3 && 3 > 2;
    printf("%d\n", result);

    result = x = y = 1;
    printf("%d\n", result);

    result = - ++x + y--;
    printf("%d\n", result);

    return 0;
}
```



```
0
1
1
-1
```

# 문제

---

- 1) 3명의 영어 점수를 입력 받으시오!
- 2) 3명의 평균 영어 점수를 구해서 printf 하시오!(float)
- 3) 3명의 점수가 모두 평균 점수에서 5점 이내에 있으면 True(1)을 print 하시오!
- 4) 각 사람의 점수를 프린트 하시오. 그런데, 그 사람의 영어점수가 평균점수보다 작다면 평균 점수를 프린트하시오(? :)

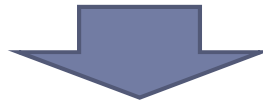


## 리포트 체크 문제(시간:20분)

---

### 리포트

- 1) 3명의 영어 점수를 입력 받으시오!
- 2) 3명의 평균 영어 점수를 구해서 printf 하시오!(float)
- 3) 3명의 점수가 모두 평균 점수에서 5점 이내에 있으면 True(1)을 print 하시오!
- 4) 각 사람의 점수를 프린트 하시오. 그런데, 그 사람의 영어점수가 평균점수보다 작다면 평균 점수를 프린트하시오(? :)



위의 프로그램을 다음과 같이 확장하시오. 20분내에 프로그램 완료 했을때 만 리포트 제출이 인정됨.

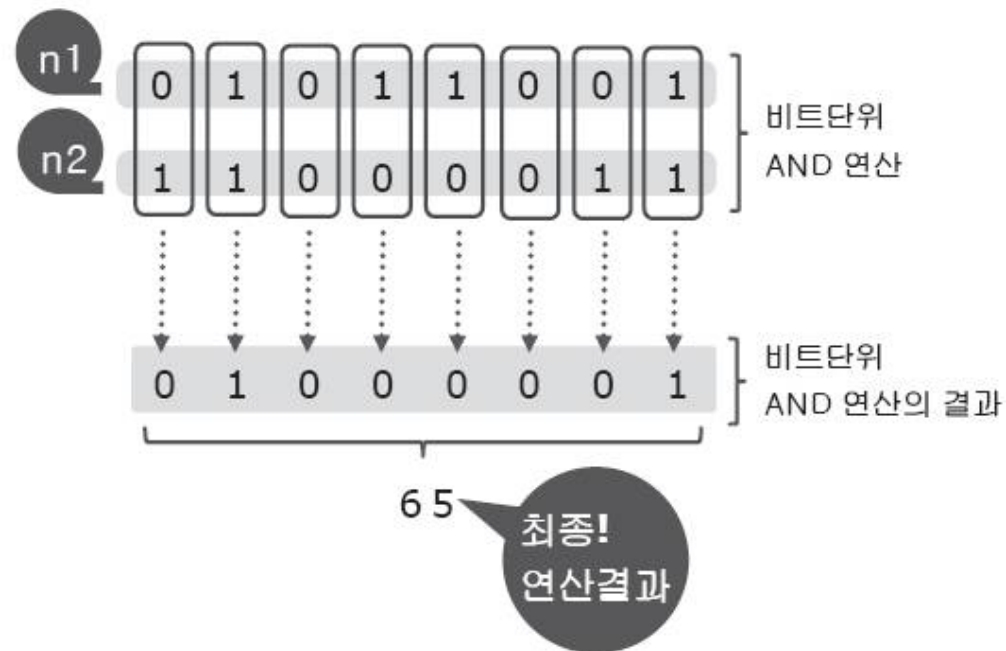
- 5) 4명의 영어점수를 입력 받도록 수정하시오.
- 6) 평균을 “소수점 2자리”까지만 printf() 하도록 프로그램을 수정하시오.
- 7) 각 사람의 점수를 프린트 하시오. 그런데, 그 사람의 영어점수가 평균점수보다 **크거나 같다면** 평균 점수를 프린트하시오



비트 연산자

# Bit 연산의 이해

- ▶ bit 연산자
  - ▶ 비트단위로 연산을 진행



## 비트 연산자(비트 이동 연산자)

| 연산자 | 연산자의 기능  | 결합방향 |
|-----|--|------|
| &   | 비트단위로 AND 연산을 한다.<br>예) num1 & num2;                                       | →    |
|     | 비트단위로 OR 연산을 한다.<br>예) num1   num2;  | →    |
| ^   | 비트단위로 XOR 연산을 한다.<br>예) num1 ^ num2;                                       | →    |
| ~   | 단항 연산자로서 피연산자의 모든 비트를 반전시킨다.<br>예) ~num;    // num은 변화 없음, 반전 결과만 반환       | ←    |
| <<  | 피연산자의 비트 열을 왼쪽으로 이동시킨다.<br>예) num<<2;    // num은 변화 없음, 두 칸 왼쪽 이동 결과만 반환   | →    |
| >>  | 피연산자의 비트 열을 오른쪽으로 이동시킨다.<br>예) num>>2;    // num은 변화 없음, 두 칸 오른쪽 이동 결과만 반환 | →    |

Shift 연산자  
left shift  
right shift

# & 연산자: 비트단위 AND

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 & num2;    // num1과 num2의 비트단위 & 연산
    printf("AND 연산의 결과: %d \n", num3);
    return 0;
}
```

AND 연산의 결과: 4

실행결과

|      |          |          |          |     |       |
|------|----------|----------|----------|-----|-------|
|      | 00000000 | 00000000 | 00000000 | 000 | 01111 |
| & 연산 | 00000000 | 00000000 | 00000000 | 000 | 10100 |
|      | 00000000 | 00000000 | 00000000 | 000 | 00100 |

- 0 & 0      0을 반환
- 0 & 1      0을 반환
- 1 & 0      0을 반환
- 1 & 1      1을 반환

& : ampersand  
(앰퍼샌드)



# | 연산자: 비트단위 OR

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 | num2;
    printf("OR 연산의 결과: %d \n", num3);
    return 0;
}
```

OR 연산의 결과: 31

실행결과

|    |          |          |          |     |       |
|----|----------|----------|----------|-----|-------|
|    | 00000000 | 00000000 | 00000000 | 000 | 01111 |
| 연산 | 00000000 | 00000000 | 00000000 | 000 | 10100 |
|    |          |          |          |     | ----- |
|    | 00000000 | 00000000 | 00000000 | 000 | 11111 |

|   |  |   |       |
|---|--|---|-------|
| 0 |  | 0 | 0을 반환 |
| 0 |  | 1 | 1을 반환 |
| 1 |  | 0 | 1을 반환 |
| 1 |  | 1 | 1을 반환 |

| : vertical bar!!  
(broken bar)

## ^ 연산자: 비트단위 XOR

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 ^ num2;
    printf("XOR 연산의 결과: %d \n", num3);
    return 0;
}
```

XOR 연산의 결과: 27

실행결과

|       |          |          |          |     |       |
|-------|----------|----------|----------|-----|-------|
|       | 00000000 | 00000000 | 00000000 | 000 | 01111 |
| ^ 연산  | 00000000 | 00000000 | 00000000 | 000 | 10100 |
| <hr/> |          |          |          |     |       |
|       | 00000000 | 00000000 | 00000000 | 000 | 11011 |

|              |       |
|--------------|-------|
| $0 \wedge 0$ | 0을 반환 |
| $0 \wedge 1$ | 1을 반환 |
| $1 \wedge 0$ | 1을 반환 |
| $1 \wedge 1$ | 0을 반환 |

^ : caret(캐럿)

## ~ 연산자

```
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = ~num1;
    printf("NOT 연산의 결과: %d \n", num2);
    return 0;
}
```

NOT 연산의 결과: -16

실행결과

- ~ 0          1을 반환
- ~ 1          0을 반환

~ 연산 전    00000000 00000000 00000000 00001111  
~ 연산 후    11111111 11111111 11111111 11110000

~ : swung dash(스윙대시)  
물결



## << 연산자: 비트의 왼쪽 이동(Left Shift)

- `num1 << num2`    `num1`의 비트 열을 `num2`칸씩 왼쪽으로 이동시킨 결과를 반환
- `8 << 2`            정수 8의 비트 열을 2칸씩 왼쪽으로 이동시킨 결과를 반환

```
int main(void)
{
    int num = 15;    // 00000000 00000000 00000000 00001111

    int result1 = num<<1;    // num의 비트 열을 왼쪽으로 1칸씩 이동
    int result2 = num<<2;    // num의 비트 열을 왼쪽으로 2칸씩 이동
    int result3 = num<<3;    // num의 비트 열을 왼쪽으로 3칸씩 이동

    printf("1칸 이동 결과: %d \n", result1);
    printf("2칸 이동 결과: %d \n", result2);
    printf("3칸 이동 결과: %d \n", result3);
    return 0;
}
```

실행결과

1칸 이동 결과: 30  
2칸 이동 결과: 60  
3칸 이동 결과: 120

00000000 00000000 00000000 00011110    // 10진수로 30  
00000000 00000000 00000000 00111100    // 10진수로 60  
00000000 00000000 00000000 01111000    // 10진수로 120

왼쪽으로 한 칸씩 이동할 때마다 정수의 값은 **두 배씩 증가한다.**

반대로 오른쪽으로 한 칸씩 이동할 때마다 정수의 값은 **반으로 줄어든다.**

## >> 연산자: 비트의 오른쪽 이동(Right Shift)

11111111 11111111 11111111 11110000 // -16



CPU에 따라서 달라지는 연산의 결과

00111111 11111111 11111111 11111100 // 0이 채워진 경우

11111111 11111111 11111111 11111100 // 1이 채워진 경우

왼쪽 비트가 0으로 채워진 음수의 경우 오른쪽으로 비트를 이동시킨 결과는 CPU에 따라서 달라진다.  
따라서 호환성이 요구되는 경우에는 >> 연산자의 사용을 제한해야 한다.

```
int main(void)
{
    int num = -16;    // 11111111 11111111 11111111 11110000
    printf("2칸 오른쪽 이동의 결과: %d \n", num>>2);
    printf("3칸 오른쪽 이동의 결과: %d \n", num>>3);
    return 0;
}
```

실행결과

2칸 오른쪽 이동의 결과: -4  
3칸 오른쪽 이동의 결과: -2

## Bit연산 활용 : 비트열의 특정부분을 출력함(Bit Mask)

### ■ 예제 15-8.c

```
1.  #include <stdio.h>
2.
3.  int main(void)
4.  {
5.      unsigned short data=0x5678; /* 0101 0110 0111 1000 */
6.
7.      unsigned short msk1=0xf000; /* 1111 0000 0000 0000 */
8.      unsigned short msk2=0x0f00; /* 0000 1111 0000 0000 */
9.      unsigned short msk3=0x00f0; /* 0000 0000 1111 0000 */
10.     unsigned short msk4=0x000f; /* 0000 0000 0000 1111 */
11.
12.     printf("result1= %#.4x \Wn", data & msk1);
13.     printf("result2= %#.4x \Wn", data & msk2);
14.     printf("result3= %#.4x \Wn", data & msk3);
15.     printf("result4= %#.4x \Wn", data & msk4);
16.     return 0;
17. }
```

비트 마스크



# 비트 마스크의 활용 사례

```
int main(void)
{
    SetKindSte(TRUE);
    SetAddOpSte(TRUE);
    SetMinOpSte(TRUE);
    ....
    ShowOperationResult(5, 2);
    ....
}
```

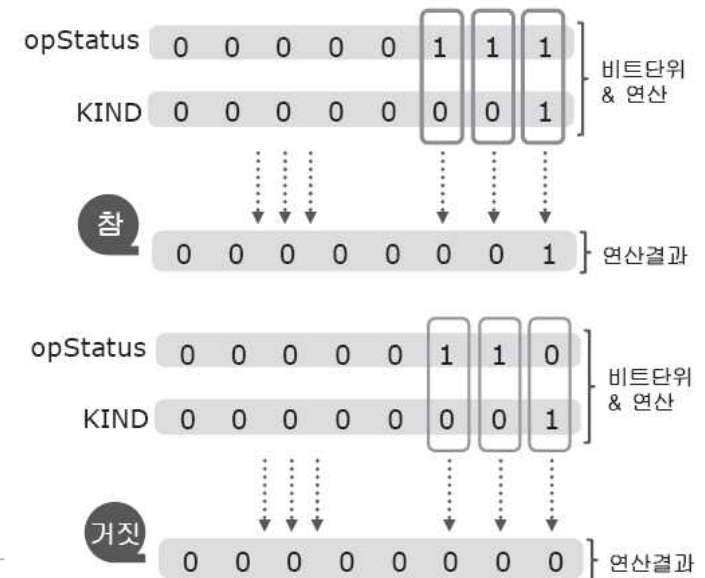
함수  
ShowOperationResult의  
실행방식 결정

```
const int KIND=1; /* 00000001 */
const int ADD=2; /* 00000010 */
const int MIN=4; /* 00000100 */
```

```
int main(void)
{
    SetShowStatus(KIND | ADD | MIN);
    ....
    ShowOperationResult(5, 2);
    ....
}
```

세 번의  
함수 호출  
대체

비트 마스크를  
이용한 비트 정보 추출



## 실습 문제

---

### ▶ 리포트 체크문제 확장

- 5) 4명의 영어점수를 입력 받도록 수정하시오.
- 6) 평균을 “소수점 2자리”까지만 printf() 하도록 프로그램을 수정하시오.
- 7) 각 사람의 점수를 프린트 하시오. 그런데, 그 사람의 영어점수가 평균점수보다 **크거나 같다면** 평균 점수를 프린트하시오



- 8) 1(left),2(right) 중 정수 하나와 16이하의 정수하나를 입력받으시오  
scanf(“%d %d”, shiftDirection, shiftCount)
- 9) 첫 번째 사람 영어점수를 shiftDirection 으로 shiftCount 만큼 shift 한 값을 출력하시오(결과 설명)
- 10) 평균점수도 9)와 같이 shift 하도록 하시오.