프로그래밍 기초

5장. 연산자와 연산식

학습목표

- **연산자의 기본 개념인 다음 용어를 이해하고 설명할 수 있다.**
  - 연산자, 피연산자, 연산식
  - 단항연산자, 이항연산자, 삼항연산자

- **다음 연산자의 사용 방법과 연산식의 결과를 알 수 있다.**
  - 산술연산자 +, –, *, /, %
  - 대입연산자 =, +=, –=. *=, /=, %=
  - 증감연산자 ++, --, 조건연산자 ?:
  - 관계연산자 >, <, >=, <=, ==, !=, 논리연산자 &&, ||, !
  - 형변환연산자 (type cast)
  - sizeof 연산자, 콤마연산자

- **연산자 우선순위에 대하여 이해하고 설명할 수 있다.**
  - 괄호, 단항, 이항, 삼항연산자의 순위
  - 산술연산자의 순위
  - 관계와 논리연산자 순위
  - 조건, 대입, 콤마연산자 순위

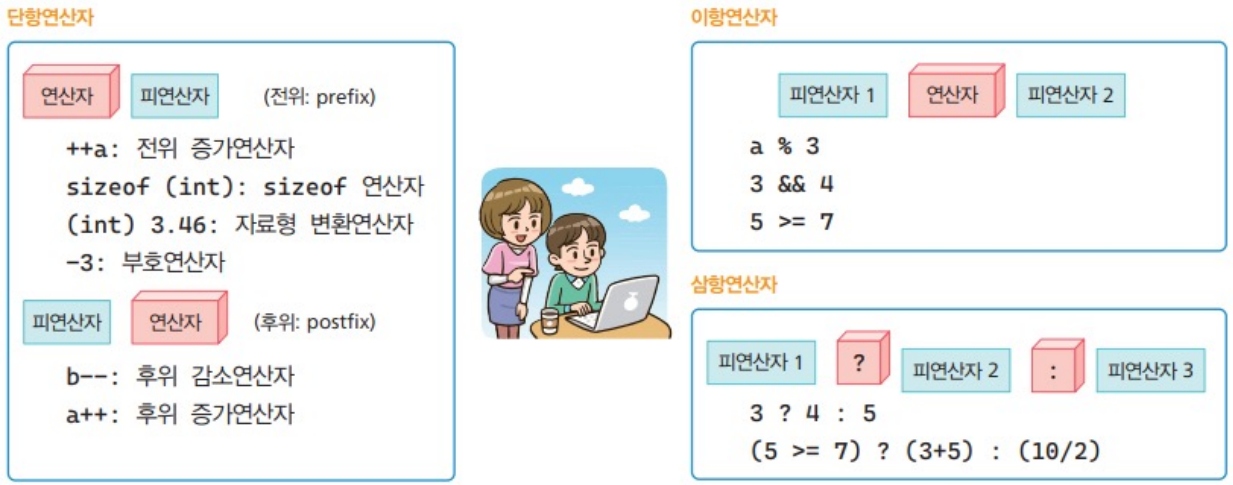
연산자와 피연산자, 연산식과 연산값

- **연산식(expression)**
  - 변수와 다양한 리터럴 상수 그리고 함수의 호출 등으로 구성되는 표현식: `3 + (4 * 5)`
  - 연산식은 항상 하나의 결과값을 가짐
- **연산자(operator)**
  - `+, -, *, /, % (modulus)`
  - 이미 정의된 연산을 수행하는 문자 또는 문자 조합 기호
- **피연산자(operand)**
  - 연산(operation)에 참여하는 변수나 상수

다양한 연산자

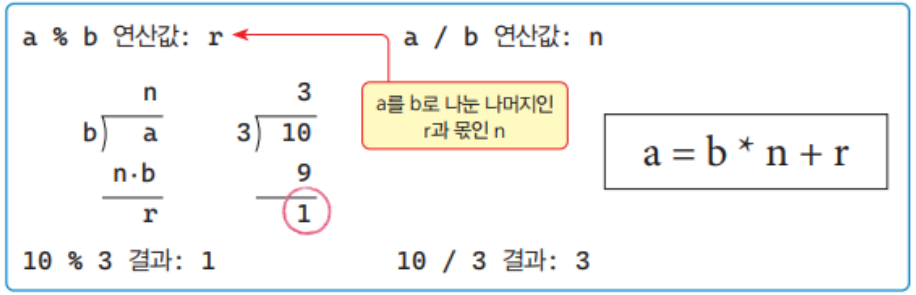
- **단(일)항(unary), 이항(binary), 삼항(ternary) 연산자**
  - 연산에 참여하는 피연산자(operand)의 개수
  - 삼항 연산자
    - ➢ 조건연산자 '? :'가 유일

- **단항연산자는 연산자의 위치에 따라**
  - ++a : 전위
    - ➢ 연산자가 앞에 있으면 전위(prefix) 연산자
  - a++: 후위
    - ➢ 연산자가 뒤에 있으면 후위(postfix) 연산자



곱하기와 나누기, 나머지 연산자 활용 (실습)

- **나머지 연산식: a % b**
  - a를 b로 나눈 나머지 값
  - 10 / 3 -> 3
  - 10 % 3 -> 1



<01arithop.c>

```c
#include <stdio.h>

int main(void)
{
    int amount = 4000 * 3 + 10000;

    printf(" 총금액 %d 원\n", amount);
    printf("오천원권 %d 개\n", amount / 5000);
    printf(" 천원권 %d 개\n", (amount % 5000) / 1000);

    return 0;
}
```

실행 결과

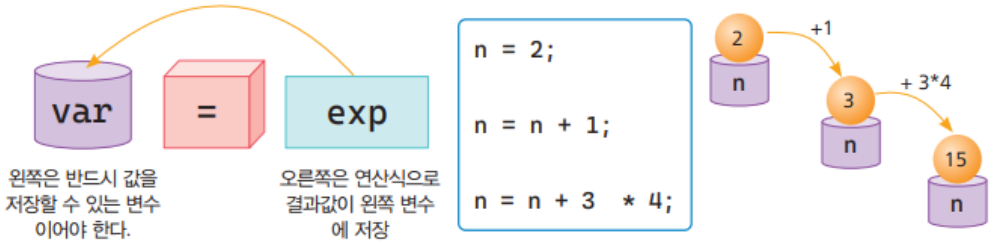
```
 총금액 22000 원
오천원권 4 개
 천원권 2 개
```

대입연산자 =

- **대입연산자(assignment operator) =**
  - 오른쪽 연산식 결과값을 왼쪽 변수에 저장하는 연산자
    - 왼쪽 부분에는 반드시 하나의 변수만이 올 수 있음

<02assignop.c>

```c
#include <stdio.h>

int main()
{
    int cred2, cred3;

    cred2 = cred3 = 0;

    printf("2학점과 3학점의 수강 과목 수를 각각 입력 >> ");
    scanf("%d %d", &cred2, &cred3);

    cred2 = 2 * cred2;
    printf("2학점 과목 총 학점: %d\n", cred2);
    printf("3학점 과목 총 학점; %d\n", cred3 = 3 * cred3);
    printf("총 학점: %d\n", cred2 + cred3);

    return 0;
}
```



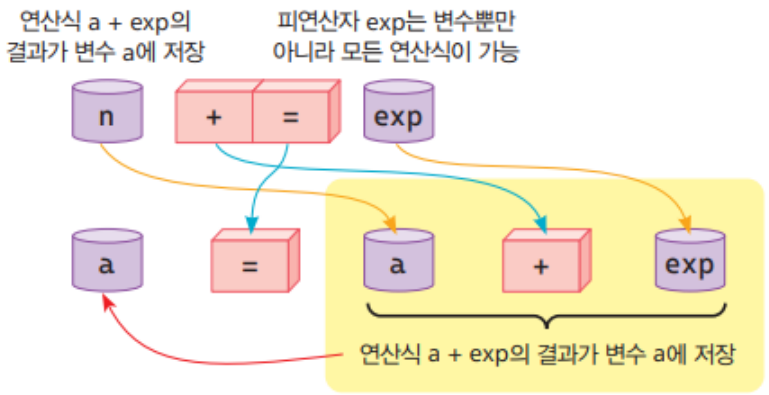
실행 결과

```
2학점과 3학점의 수강 과목 수를 각각 입력 >> 2 4
2학점 과목 총 학점: 4
3학점 과목 총 학점; 12
총 학점: 16
```

축약 대입연산자

- **+= -= *= /= %=**

연산	축약 대입연산
op1 = op1 + op2	op1 += op2
op1 = op1 – op2	op1 -= op2
op1 = op1 * op2	op1 *= op2
op1 = op1 / op2	op1 /= op2
op1 = op1 % op2	op1 %= op2



<03compoundassign.c>

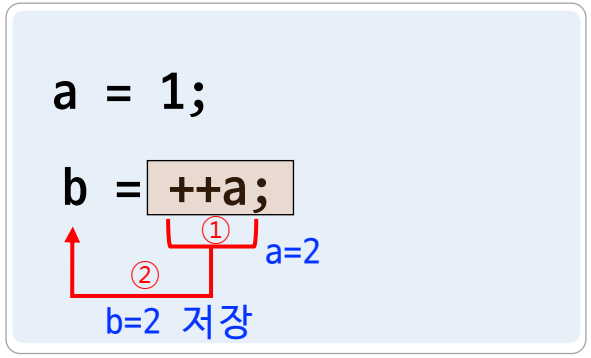
```c
#include <stdio.h>

int main()
{
    int x=0, y=0;

    printf("두 정수를 입력 >> ");
    scanf("%d %d", &x, &y);

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

    return 0;
}
```

실행 결과

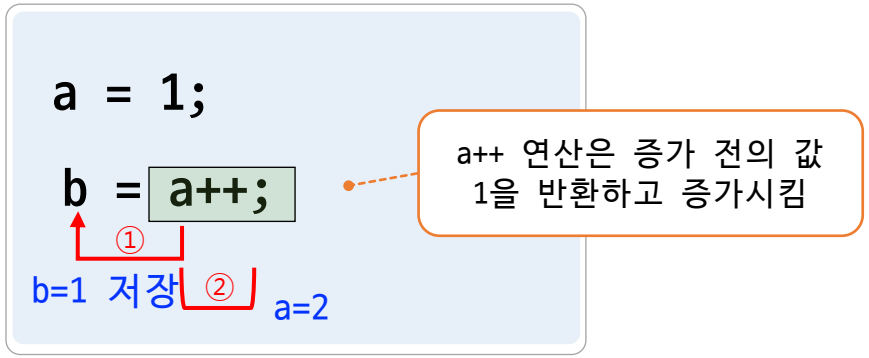
```
두 정수를 입력 >> 10 20
30
30 20
10
10 20
```

증감 연산자 ++ --

- **증감 연산자(++, --)**
  - 변수의 값을 1 증가 시키거나 1 감소 시킴

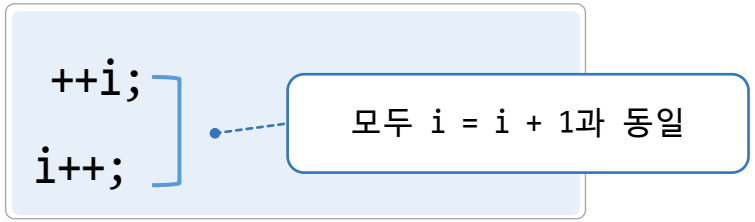


전위 연산자



후위 연산자

- **변수 단독으로 증감 연산자 사용**
  - 전위, 후위 연산자의 결과는 동일



전위 연산자	내용	후위 연산자	내용
++a	a를 1 증가하고 증가된 값을 반환	a++	a값을 반환하고, 나중에 1증가
--a	a를 1 감소하고 감소된 값 반환	a--	a값을 반환하고, 나중에 1 감소

증가연산자 ++와 감소연산자 -- (실습)

<04incdecop.c>

```c
#include <stdio.h>

int main(void)
{
    int m = 1, n = 5;

    printf("%d %d\n", m++, ++n); //1 6
    printf("%d %d\n", m, n);     //2 6
    printf("%d %d\n", m--, --n); //2 5
    printf("%d %d\n", m, n);     //1 5

    return 0;
}
```

실행 결과

```
1 6
2 6
2 5
1 5
```

```c
int n = 10;

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

출력: 10, 11

```c
int n = 10;

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

출력: 11, 11

```c
int n = 10;

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

출력: 10, 9

```c
int n = 10;

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

출력: 9, 9

Lab. 연산자 /와 %를 사용한 지폐 계산 방법 (실습)

- **식비 지불 금액이 78,900원**
  - 오만원권과 만원권, 그리고 오천원, 천원 지폐를 몇 개씩 지불하고 나머지는 얼마를 내면 될까?

<lab1pay.c>

```c
#include <stdio.h>

int main(void)
{
    int amount;
    printf("총 금액 입력 >> ");
    scanf("%d", &amount);
    printf("계산 금액: %d\n", amount);

    int cnt50000 = amount / 50000;
    int changes = amount % 50000;
    printf("50,000Won: %d개\n", cnt50000);

    int cnt10000 = changes / 10000;
    changes %= 10000; //changes = changes % 10000;
    printf("10,000Won: %d개\n", cnt10000);

    int cnt5000 = changes / 5000;
    changes %= 5000; //changes = changes % 5000;
    printf("5,000Won: %d개\n", cnt5000);

    int cnt1000 = changes / 1000;
    changes %= 1000; //changes = changes % 1000;
    printf("1,000Won: %d개\n", cnt1000);
    printf("나마저: %d개\n", changes);
    return 0;
}
```

실행 결과

```
총 금액 입력 >> 78900
계산 금액: 78900
50,000Won: 1개
 10,000Won: 2개
 5,000Won: 1개
 1,000Won: 3개
나머지: 900
```

관계 연산자

- **관계연산자는 두 피연산자의 크기를 비교하기 위한 연산자**
  - 비교 결과가 참(true)이면 1, 거짓(false)이면 0

<05relationop.c>

```c
#include <stdio.h>

int main(void)
{
    int speed = 80;

    printf("(60 <= speed): %d\n", (60 <= speed));
    printf("(60 > speed): %d\n", (60 > speed));

    printf("(a > b): %d\n", ('a' > 'b'));
    printf("(Z <= a): %d\n", ('Z' <= 'a'));
    printf("(4 == 4.0): %d\n", (4 == 4.0));
    printf("(4.0F != 4.0): %d\n", (4.0F != 4.0));

    return 0;
}
```

연산자	연산식	의미	예제	연산(결과)값
>	x > y	x가 y보다 큰가?	3 > 5	0(거짓)
>=	x >= y	x가 y보다 크거나 같은가?	5-4 >= 0	1(참)
<	x < y	x가 y보다 작은가?	'a' < 'b'	1(참)
<=	x <= y	x가 y보다 작거나 같은가?	3.43 <= 5.862	1(참)
!=	x != y	x와 y가 다른가?	5-4 != 3/2	0(거짓)
==	x == y	x가 y가 같은가?	'%' == 'A'	0(거짓)

실행 결과

```
(60 <= speed): 1
(60 > speed): 0
(a > b): 0
(Z <= a): 1
(4 == 4.0): 1
(4.0F != 4.0): 0
```

ASCII Code Table에서
Z=90, a=97로 정의되어 있음

논리연산자

- **논리연산자 &&(and), ||(or), !(not)을 제공**
  - 결과가 참(true)이면 1 거짓(false)이면 0을 반환
    - 0, 0.0, '\0'은 거짓을 의미
    - 0이 아닌 모든 정수와 실수, 그리고 널 (null) 문자 '\0'가 아닌 모든 문자와 문자열은 모두 참을 의미

<06logicop.c>

```c
#include <stdio.h>

int main(void)
{
    double grade = 4.21;

    printf("%d ", (4.0 < grade) && (grade <= 5)); // 1
    printf("%d ", 0.0 || (4.0 > grade));          // 0
    printf("%d\n", (4.2 < grade) || !0.0);        // 1

    printf("%d ", 'a' && 3.5);                    // 1
    printf("%d ", '\0' || "C");                   // 1
    printf("%d\n", "java" && '\0');               // 0

    return 0;
}
```

x	y	x && y	x || y	!x
0(거짓)	0(거짓)	0	0	1
0(거짓)	0이 아닌 값(참)	0	1	1
0이 아닌 값(참)	0(거짓)	0	1	0
0이 아닌 값(참)	0이 아닌 값(참)	1	1	0

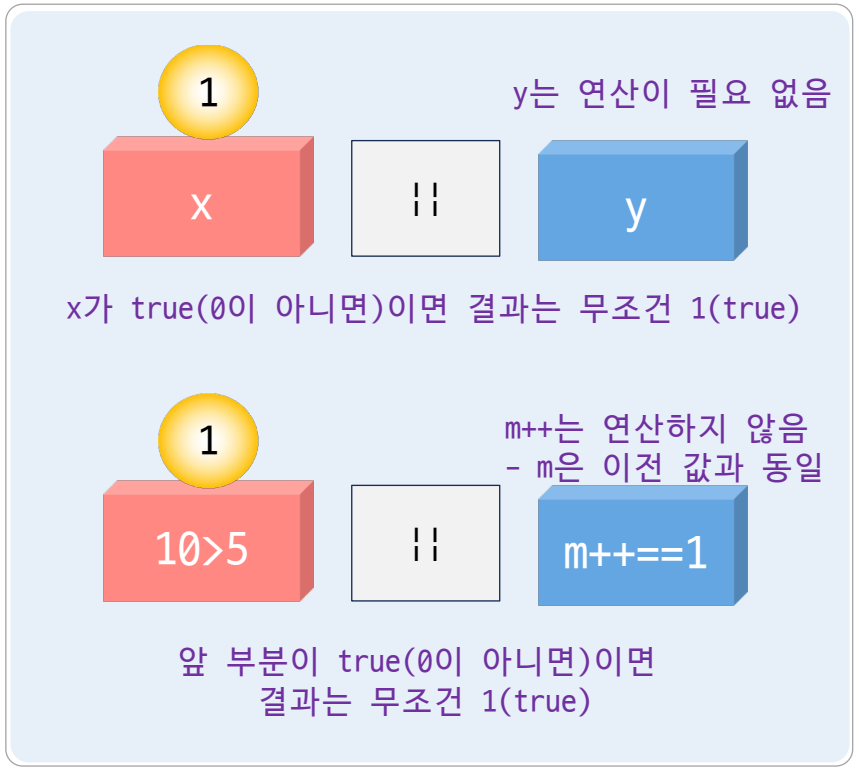
- 'a': 97이므로 0이 아님(true)
- 3.5: 0이 아님(true)
- true && true -> true

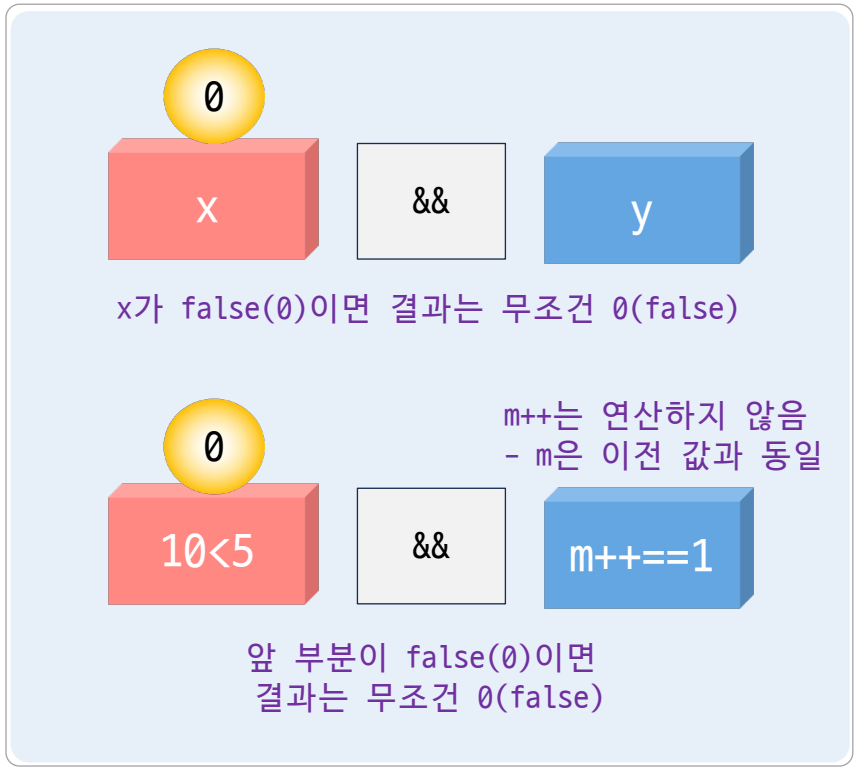
실행 결과

```
1 0 1
1 1 0
```

논리 연산자 단축 평가

- **논리연산자 &&와 ||**
  - 피연산자 두 개 중에서 왼쪽 피연산자만으로 논리연산 결과가 결정된다면
    - ➢ 오른쪽 피연산자는 평가하지 않는 방식





단축 평가 예제

- 논리연산자 &&와 ||

<07shorteval.c>

```c
#include <stdio.h>

int main(void)
{
    int amount = 0;
    int coupons = 10; //각각 10 이상과 10 미만을 입력

    printf("총 금액 >> ");
    scanf("%d", &amount); // 각각 10000원 이상과 미만을 입력


    int sale = (amount >= 10000) && (coupons++ >= 10);
    printf("할인: %d, 쿠폰 수: %d\n", sale, coupons);

    return 0;
}
```

(amount >= 10000)가 참이면
(coupons++ >= 10)을 수행

실행 결과 1

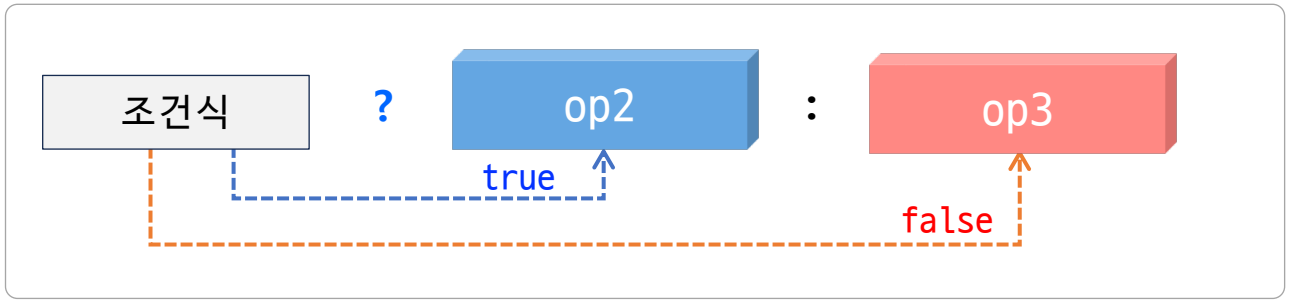
```
총 금액 >> 9000
할인: 0, 쿠폰 수: 10
```

실행 결과 2

```
총 금액 >> 15000
할인: 1, 쿠폰 수: 11
```

조건연산자 ?:

- **조건 연산자: 3항 연산자**
  - (조건식) ? (참) : (거짓)
    - 조건식이 참(true)이면 콜론(:) 앞의 op2를 수행
    - 조건식이 거짓(false)이면 콜론(:) 뒤의 op3를 수행



  - 3항 연산자 활용 예제

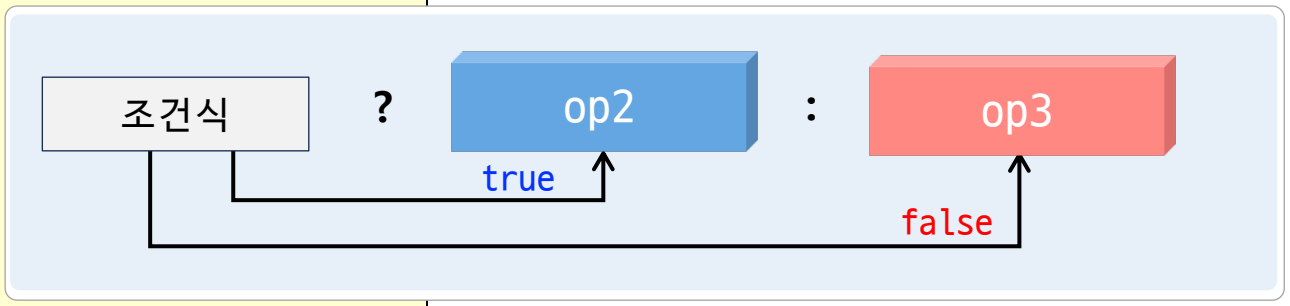
```
max = (a > b) ? a : b;  // 최대값 반환 조건연산
min = (a < b) ? a : b;  // 최소값 반환 조건연산
abs = (a > 0) ? a : -a; // 절대값 반환 조건연산
```

조건연산자 (실습)

- **삼항 연산자: (조건) ? (참) : (거짓)**
  - 조건에 따라 주어진 피연산자가 결과값이 되는 삼항연산자

<08condop.c>

```c
#include <stdio.h>

int main(void)
{
    int a = 0, b = 0;
    printf("두 정수 입력 >> ");
    scanf("%d%d", &a, &b);

    printf("최대값: %d ", (a > b) ? a : b);
    printf("최소값: %d\n", (a < b) ? a : b);
    printf("절대값: %d ", (a > 0) ? a : -a);
    printf("절대값: %d\n", (b > 0) ? b : -b);

    ((a % 2) == 0) ? printf("짝수") : printf("홀수");
    printf("%s\n", ((b % 2) == 0) ? "짝수" : "홀수");

    return 0;
}
```



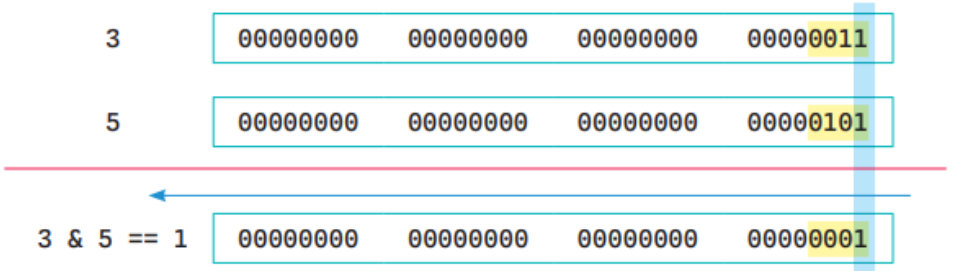
실행 결과

```
두 정수 입력 >> 8 -9
최대값: 8 최소값: -9
절대값: 8 절대값: 9
짝수 홀수
```

비트 논리 연산자

비트 논리 연산자 (&, ¦, ^, ~)

연산자	연산자 이름	사용	의미
&	비트 AND	op1 & op2	비트가 모두 1이면 결과는 1, 아니면 0
|	비트 OR	op1 | op2	비트가 적어도 하나 1이면 결과는 1, 아니면 0
^	비트 배타적 OR(XOR)	op1 ^ op2	비트가 서로 다르면 결과는 1, 같으면 0
~	비트 NOT(Negation) 또는 보수(complement)	~op1	비트가 0이면 결과는 1, 0이면 1



- 보수 연산자(bitwise complement operator): ~
  - 각 비트에서 0은 1로, 1은 0으로 변환

피연산자		보수 연산	
수	비트표현(2진수)	보수 연산 결과	10진수
1	00000000 00000000 00000000 00000001	11111111 11111111 11111111 11111110	~1 = -2
4	00000000 00000000 00000000 00000100	11111111 11111111 11111111 11111011	~4 = -5

XOR 연산

```
3 ^ 4 = 7          3 ^ 5 = 6

  0011               0011
^ 0100             ^ 0101
-------            -------
  0111               0110
```

정수의 음수 표현 방법

- 양의 정수 a에서 음수인 –a의 비트 표현은 2의 보수 표현 사용: (~a + 1)

비트 연산자 & | ^ ~

<09bitop.c>

```c
#include <stdio.h>

int main(void)
{
    int x = 15; // 1111

    printf("%8x\n", -1); // ff: 1111 1111
    printf("%3d\n", 10 & -1);
    printf("%3d\n\n", 10 | 0);

    printf("%3d %08x\n", x, x); // 1111
    printf("%3d %08x\n", 1, x & 1); // 1111 & 0001
    printf("%3d %08x\n", 15, x | 1); // 1111 | 0001
    printf("%3d %08x\n", 14, x ^ 1); // XOR: 1111 ^ 0001 -> 1110 (0xe)
    printf("%3d %08x\n", ~x, ~x); //-16

    return 0;
}
```

x=15: 0 0 0 0 1 1 1 1

~x(메모리): 1 1 1 1 0 0 0 0

~x 음수 해석 과정 (부호 있는 정수로 출력)

1의 보수: 1 0 0 0 1 1 1 1

2의 보수= -16: 1 0 0 1 0 0 0 0

0xff 해석 과정 (부호 있는 정수로 출력)

-1이 실제 메모리에 저장된 값

1 1 1 1 1 1 1 1 — 0xff

부호 — 1의 보수 계산

1 0 0 0 0 0 0 0

1의 보수 계산+1 -> 2의 보수

1 0 0 0 0 0 0 1

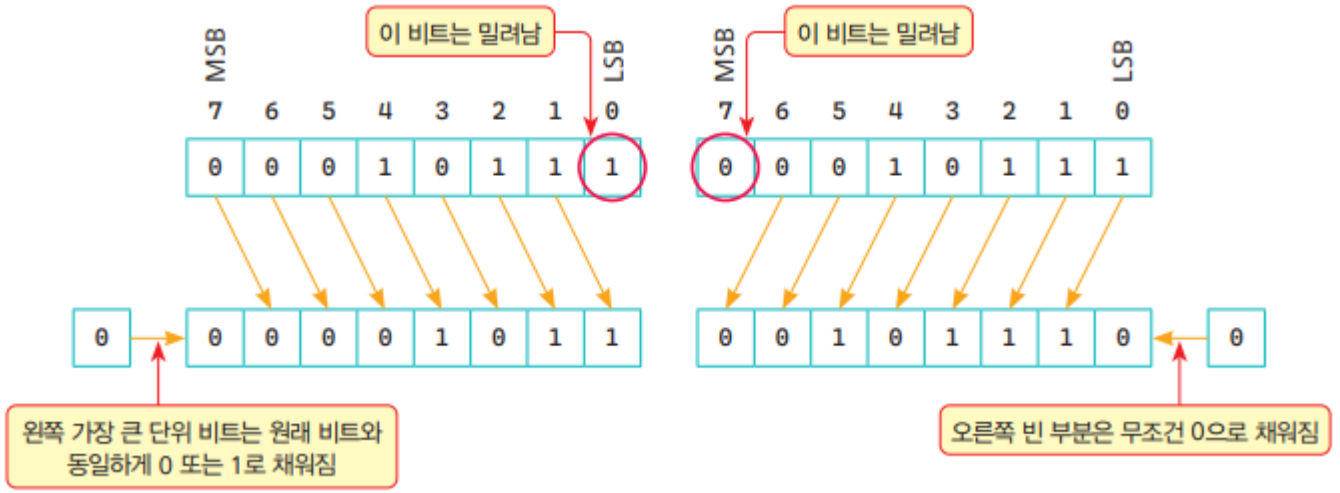
10진수로 출력할 때: -1

실행 결과

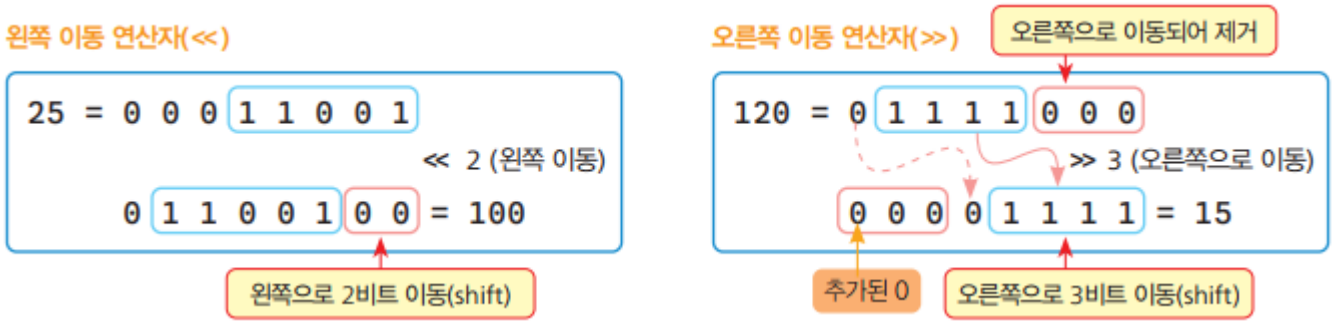
```
ffffffff
 10
 10

 15 0000000f
  1 00000001
 15 0000000f
 14 0000000e
-16 fffffff0
```

비트 이동 연산자

- **비트 이동 연산(>>, <<)**
  - 왼쪽이나 오른쪽으로 비트를 이동
- **Shift right: >>**
  - 원래의 부호비트에 따라 0 또는 1이 채워짐
  - 나누기 2와 동일한 결과
- **Shift left: <<**
  - 오른쪽 빈 자리는 무조건 0으로 채움
  - 곱하기 2와 동일한 결과



연산자	이름	사용	연산 방법	새로 채워지는 비트
>>	shift left	op1 >> op2	op1을 오른쪽으로 op2 비트만큼 이동	가장 왼쪽 비트인 부호 비트는 원래의 부호 비트로 채움
<<	shift right	op1 << op2	op1을 왼쪽으로 op2 비트만큼 이동	가장 오른쪽 비트를 모두 0으로 채움



예제: 비트 이동 연산자

<10shiftop.c>

```c
#include <stdio.h>

int main(void)
{
    int x = 0xffff; //정수 65536

    printf("%6d %08x\n", x, x); // 0000 1111(f) 1111(f) 1111(f) 1111(f)
    printf("%6d %08x\n", x >> 1, x >> 1); // 0000 0111(7) 1111(f) 1111(f) 1111(f)
    printf("%6d %08x\n", x >> 2, x >> 2); // 0000 0011(3) 1111(f) 1111(f) 1111(f)
    printf("%6d %08x\n", x >> 3, x >> 3); // 0000 0001(1) 1111(f) 1111(f) 1111(f)

    printf("%6d %08x\n", x << 1, x << 1); // 0001(1) 1111(f) 1111(f) 1111(f) 1110(e)
    printf("%6d %08x\n", x << 2, x << 2); // 0011(3) 1111(f) 1111(f) 1111(f) 1100(c)

    return 0;
}
```

실행 결과

```
 65535 0000ffff
 32767 00007fff
 16383 00003fff
  8191 00001fff
131070 0001fffe
262140 0003fffc
```

정수에서 오른쪽 n번째 비트 값 알기

- **비트 연산자를 이용하여 정수에서 오른쪽 n번째 비트 값 알기**

TIP 비트 연산자를 이용하여 정수에서 오른쪽 n번째 비트 값 알기

임의 정수 x의 비트 연산 x&1의 결과는 0 또는 1이다. 즉 결과는 정수 x의 가장 오른쪽 비트 값이 0이면 0, 1이면 1이된다. 그렇다면 어느 정수에서 오른쪽 n번째 비트 값을 알 수 있는 방법을 생각해 보자. 비트 연산 x >> (n-1)은 x의 오른쪽 n번째 비트를 가장 오른쪽으로 이동시킨다. 그러므로 비트 연산 (x >> (n-1)) & 1의 결과가 바로 정수에서 오른쪽 n번째 비트 값이라는 것이라는 알 수 있다. 이와 같은 비트 연산을 이용하면 정수를 이진수로 표현할 수 있다.

```
int x = 0x2f;

printf("%d", x >> 7 & 1); //8 번째 자리
printf("%d", x >> 6 & 1); //7 번째 자리
printf("%d", x >> 5 & 1); //6 번째 자리
printf("%d", x >> 4 & 1); //5 번째 자리
```

Lab 비트 XOR 연산자 ^를 사용한 암호화와 복호화 (추가)

<lab2encryption.c>

```c
#include <stdio.h>

int main(void)
{
    int key = 12345678; //키로 사용할 정수 하나를 저장

    int origin;
    printf("ID로 사용할 8자리의 정수를 입력하세요 >> ");
    scanf("%d", &origin);

    int encode = origin ^ key; //ID를 암호화시켜 저장
    printf("입력한 ID: %d\n", origin);
    printf("암호화하여 저장된 ID: %d\n", encode);

    int input;
    printf("로그인할 ID 입력하세요 >> ");
    scanf("%d", &input);

    int result = encode ^ key; //키로 암호화된 것을 복호화
    printf("로그인 성공 여부: %d\n", input == result);

    return 0;
}
```

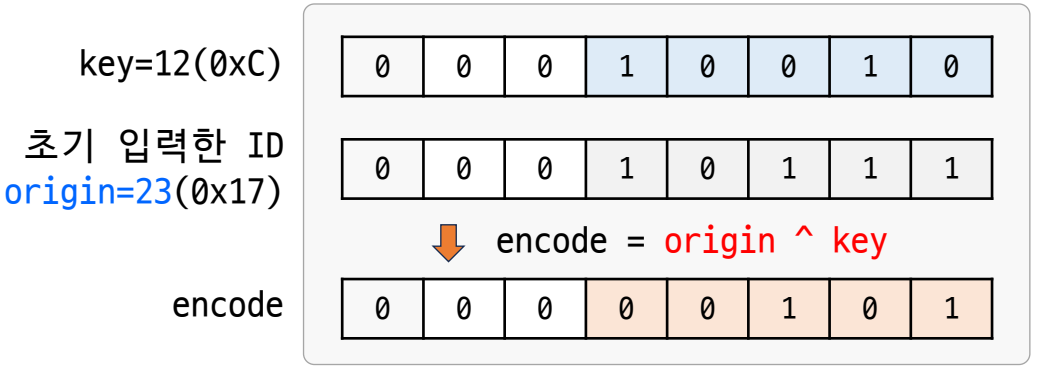
실행 결과 #1

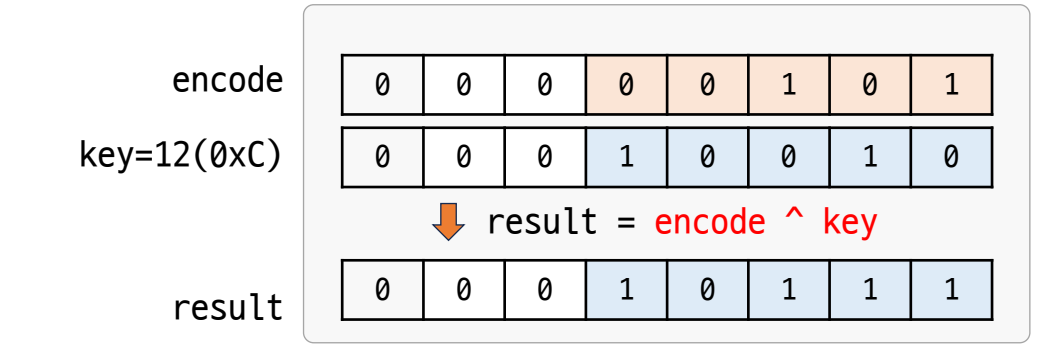
```
ID로 사용할 8자리의 정수를 입력하세요 >> 23455678
입력한 ID: 23455678
암호화하여 저장된 ID: 31033072
로그인할 ID 입력하세요 >> 23455678
로그인 성공 여부: 1
```

실행 결과 #2

```
ID로 사용할 8자리의 정수를 입력하세요 >> 23456789
입력한 ID: 23456789
암호화하여 저장된 ID: 31034715
로그인할 ID 입력하세요 >> 13456789
로그인 성공 여부: 0
```

Lab 비트 XOR 연산자 ^를 사용한 암호화와 복호화 실행 결과





로그인할 ID input=23(0x17)

로그인 성공 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

input == result

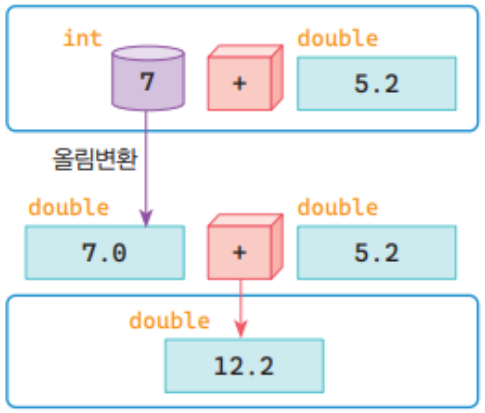
로그인할 ID input=24(0x18)

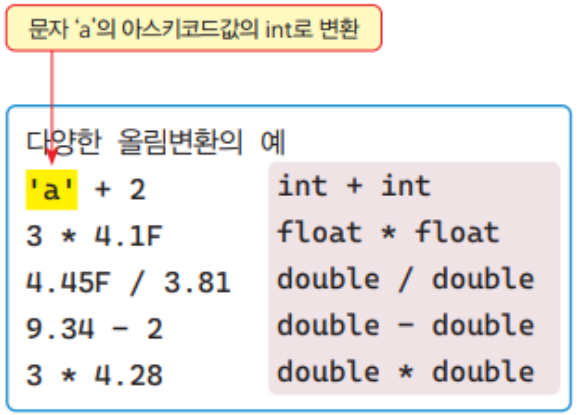
로그인 실패 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

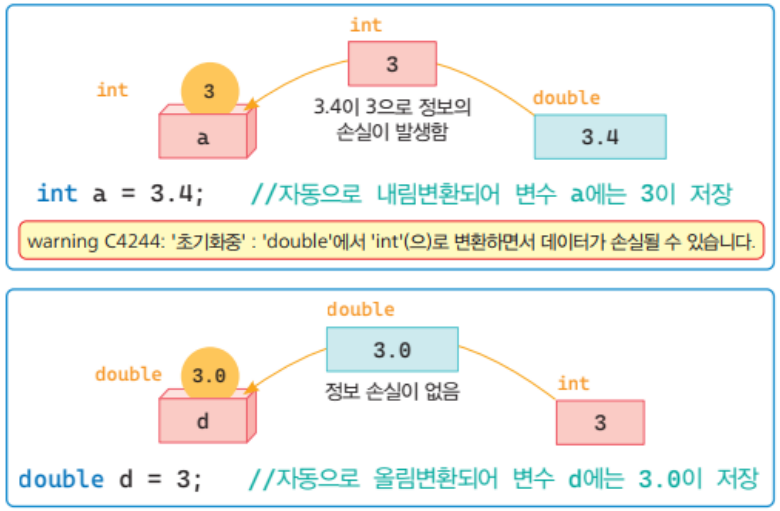
input != result

자료형 변환

- **자료형 변환 (type casting, type conversion)**
  - 자료형의 표현 방식을 바꾸는 것
- **올림변환**
  - 작은 자료형(int)에서 보다 큰 자료형(double)으로 형 변환
  - 정보 손실이 없고, 컴파일러에 의해 자동으로 수행
- **내림변환**
  - 큰 자료형(double)에서 보다 작은 자료형(int)으로 형 변환
  - 프로그래머의 명시적 형변환이 필요



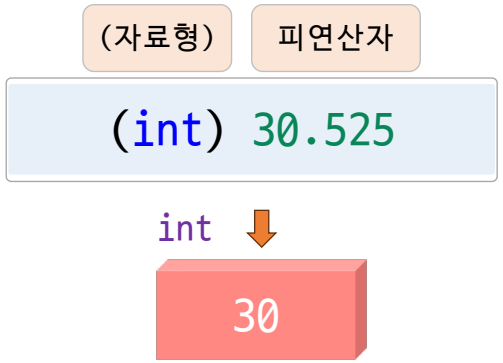




형변환 연산자

- **자료형 변환 구분 방식**
  - 명시적(강제) 형변환
    - ➢ 직접 형 변환 연산자를 사용하는 방식
  - 묵시적(자동) 형변환
    - ➢ 컴파일러가 알아서 자동으로 수행하는 방식

명시적 형변환




```
double result = (double)7 / 2;
```

double result

7.0 / 2;

3.5

형변환 연산자 (실습)

- **명시적 형변환:** (int)30.525

<11typecast.c>

```c
#include <stdio.h>

int main(void)
{
    int a = 7.8; //자동으로 내림 변환되어 변수 a에는 7이 저장
    double b = 5; //자동으로 올림 변환되어 변수 b에는 5.0이 저장

    printf("%d, %f ", a, b);
    printf("%d, %f ", (int)3.56, (double)3);
    printf("%f, %d\n", 3.56 + 7.87, (int)(3.56 + 7.87));

    printf("%d, %f, %f\n", 5 / 2, (double)5 / 2, (double) (5 / 2));

    return 0;
}
```

Warning: implicit conversion from 'double' to 'int'

명시적 형변환:
type casting

```
7, 5.000000
3, 3.000000
 11.430000, 11
2, 2.500000, 2.000000
```

sizeof 연산자

- **sizeof 3.14 또는 sizeof(3.14)**
  - 연산값 또는 자료형의 저장 장소의 크기를 구하는 연산자
    - ➢ 바이트 단위의 정수
  - 피연산자 앞에 위치하는 전위 연산자
  - 피연산자가 int와 같은 자료형인 경우 반드시 괄호를 사용: sizeof(int)
  - 피연산자가 상수나 변수 또는 연산식이면 괄호는 생략 가능

- **반환 값 자료형**
  - 자료형 size_t
  - printf()에서 형식제어문자 %zu로 출력

```
size_t sz = sizeof(short);
printf("%zu\n", sz);
```

- **콤마 연산자 ,**
  - 왼쪽과 오른쪽 연산식을 순차적으로 계산
  - 결과값은 가장 오른쪽에서 수행한 연산의 결과
  - 연산의 우선순위가 가장 낮아 대입연산자보다 나중에 계산

```
x = 3 + 4, 2 * 3; // (x = 3 + 4), 2*3;
x = (3 + 4, 2 * 3);
```

연산자 sizeof와 콤마 연산자의 활용

<12sizeofcomma.c>

```c
#include <stdio.h>

int main(void)
{
    int a, x;
    a = x = 0;

    x = 3 + 4, 2 * 3; // (x = 3 + 4), 2*3;
    printf("x = %d\n", x);

    x = (3 + 4, 2 * 3); // x = 2 * 3;
    printf("x = %d\n", x);

    int byte = sizeof(double);
    printf("double 형: %d bytes, %d bits\n", byte, byte * 8);

    int bit = (byte = sizeof a, byte * 8);
    printf("int 형: %d bytes, %d bits\n", byte, bit);

    size_t sz = sizeof(short);
    printf("%zu\n", sz); // printf("%zu\n", sizeof(short));
    return 0;
}
```

실행 결과

```
x = 7
x = 6
double 형: 8 bytes, 64 bits
int 형: 4 bytes, 32 bits
2
```

연산자 우선순위와 결합성

- **연산 규칙**
  - 첫 번째 규칙: 괄호가 있으면 먼저 계산
  - 두 번째 규칙: 연산의 우선순위(priority)
    - *, /, % 연산은 +, - 보다 우선
  - 세 번째 규칙: 동일한 우선순위인 경우
    - **왼쪽부터 오른쪽**으로 차례로 계산
    - 제곱승과 같은 정해진 연산은 오른쪽에서 왼쪽으로 차례로 계산

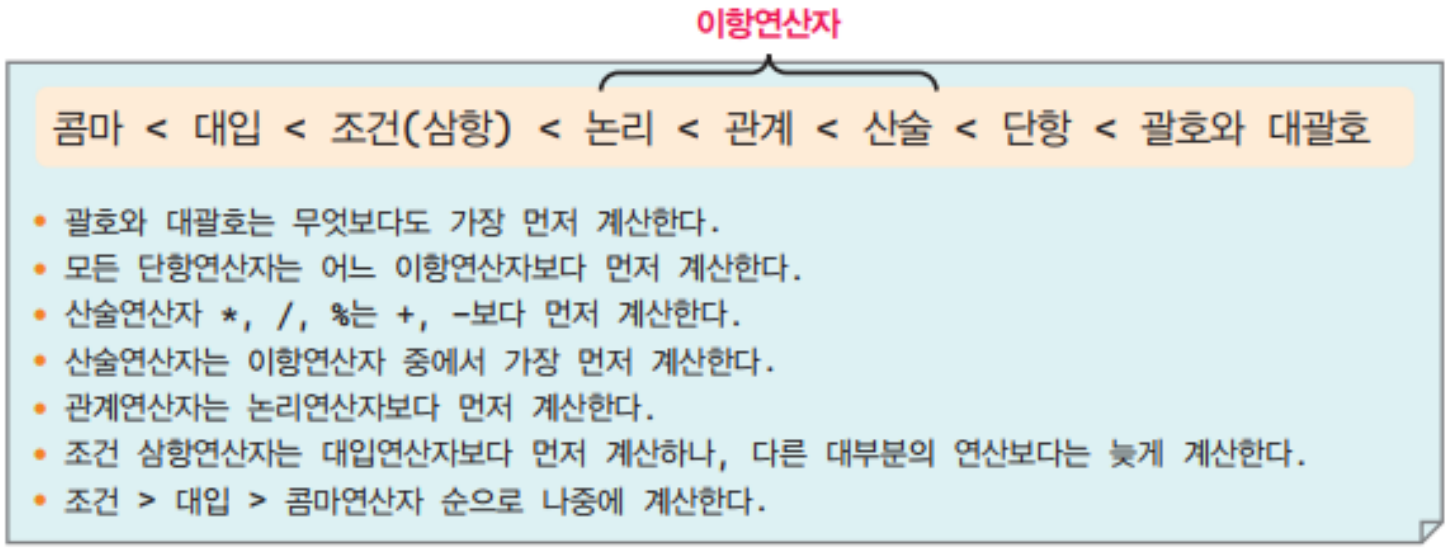


표 5-9 C 언어의 연산자 우선순위

우선순위	연산자	설명	분류	결합성(계산방향)
1	( ) [ ] . -> a++ a--	함수 호출 및 우선 지정 인덱스 필드(유니온) 멤버 지정 필드(유니온)포인터 멤버 지정 후위 증가, 후위 감소	단항	-> (좌에서 우로)
2	++a --a ! ~ sizeof - + & *	전위 증가, 전위 감소 논리 NOT, 비트 NOT(보수) 변수, 자료형, 상수의 바이트 단위 크기 음수 부호, 양수 부호 주소 간접, 역참조		<- (우에서 좌로)
3	(형변환)	형변환		
4	* / %	곱하기 나누기 나머지	산술	-> (좌에서 우로)
5	+ -	더하기 빼기		-> (좌에서 우로)
6	<< >>	비트 이동	이동	-> (좌에서 우로)
7	< > <= >=	대소 비교	관계	-> (좌에서 우로)
8	== !=	동등 비교		-> (좌에서 우로)
9	&	비트 AND 또는 논리 AND	비트	-> (좌에서 우로)
10	^	비트 XOR 또는 논리 XOR		-> (좌에서 우로)
11	|	비트 OR 또는 논리 OR		-> (좌에서 우로)
12	&&	논리 AND(단락 계산)	논리	-> (좌에서 우로)
13	||	논리 OR(단락 계산)		-> (좌에서 우로)
14	? :	조건	조건	<- (우에서 좌로)
15	= += -= *= /= %= <<= >>= &= |= ^=	대입	대입	<- (우에서 좌로)
16	,	콤마	콤마	-> (좌에서 우로)

연산자 우선순위에 위한 계산

<13oppriority.c>

```c
#include <stdio.h>

int main(void)
{
    int speed = 90;
    int x = 1, y = 2, z = 3;

    printf("%d ", 60 <= speed && speed <= 80 + 20); //산술 > 관계 > 논리
    printf("%d ", (60 <= speed) && (speed <= (80 + 20)));

    printf("%d ", x % 2 == 0 ? y + z : y * z); //산술 > 관계 > 조건
    printf("%d ", (x % 2 == 0) ? (y + z) : (y * z));

    printf("%d ", speed += ++x && y - 2); //단항++ > 산술 > 논리 > 대입
    printf("%d\n", speed += ( (++x) && (y - 2) ));

    return 0;
}
```

실행 결과

```
1 1 6 6 90 90
```

연산자의 결합성에 따른 계산 순서 확인

<14opassociation.c>

```c
#include <stdio.h>

int main(void)
{
    int m = 5, n = 10;
    printf("%d\n", m += n /= 3); //우측에서 좌측으로 결합, m = m + n <- (n = n / 3)
    printf("%d %d\n", m, n); //8, 3

    //우측에서 좌측으로 결합
    printf("%d ", 3 > 4 ? 3 - 4 : 3 > 4 ? 3 + 4 : 3 * 4); //12
    printf("%d\n", 3 > 4 ? 3 - 4 : (3 > 4 ? 3 + 4 : 3 * 4)); //위와 같은 12

    printf("%d ", 10 * 3 / 2); //좌측에서 우측으로 결합, 15
    printf("%d\n", 10 * (3 / 2)); //우측(괄호)에서 좌측으로 결합, 10

    return 0;
}
```

절대 사용하지 말 것

```
n = n / 3;
m = m + n;
```

동일 우선 순위: 좌->우

명확하게 괄호를 사용

실행 결과

```
8
8 3
12 12
15 10
```

Lab 섭씨 온도를 화씨 온도로 변환해 출력

- 섭씨(C) 온도를 화씨 온도(F)로 변환하는 식

$$F = \frac{9}{5}C + 32$$

Lab 5-3 lab3celtofah.c 난이도: ★

```
01  #define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
02  #include <stdio.h>
03
04  int main(void)
05  {
06      double celsius, fahrenheit;
07      printf("변환할 섭씨 온도를 입력 >> ");
08      scanf("%lf", &celsius);
09
10      [                                    ]
11      printf("섭씨 %.2f: 화씨 %.2f\n", [                    ]);
12
13      return 0;
14  }
```

정답

```
10      fahrenheit = (9.0 / 5.0) * celsius + 32.0;
11      printf("섭씨 %.2f: 화씨 %.2f\n", celsius, fahrenheit);
```

Questions?