Operators

Section 1 산술 연산자
Section 2 관계 연산자
Section 3 논리 연산자
Section 4 비트 연산자
Section 5 연산자 우선순위

1 Arithmetic Operators

기본 산술 연산자

- **5칙 연산 + 대입**

표 4-1 산술 연산자

연산자	명칭	사용 예	설명
=	대입 연산자	a = 3	정수 3을 a에 대입한다.
+	더하기	a = 5 + 3	정수 5와 3을 더한 값을 a에 대입한다.
−	빼기	a = 5 − 3	정수 5에서 3을 뺀 값을 a에 대입한다.
*	곱하기	a = 5 * 3	정수 5와 3을 곱한 값을 a에 대입한다.
/	나누기	a = 5 / 3	정수 5를 3으로 나눈 값을 a에 대입한다.
%	나머지값	a = 5 % 3	정수 5를 3으로 나눈 뒤 나머지 값을 a에 대입한다.

- **특별한 기본 산술 연산: /, %**
  - Operand가 모두 정수일 때 → %d
  - Operand 중 실수가 있을 때 → %f

1 Arithmetic Operators
기본 산술 연산자

- **기본 산술 연산자의 우선순위**
  - 사람의 수 체계에서의 우선 선위와 같다
  - 괄호 >> 곱셈과 나눗셈 >> 덧셈과 뺄셈 >> 대입
  - 간단한 연산자 우선순위
    - 덧셈, 뺄셈은 연산자 우선순위가 동일하므로 아래 두 식의 결과는 같다.

```
❶ result1 = (a + b) - c;
❷ result1 = a + (b - c);
```

    - 연산자 우선순위에 의해 [응용 4-2]의 12행 결과는 아래의 ❷번 식으로 나옴.
      (* 이 + 보다 우선순위가 높음)

```
❶ result1 = (a + b) * c; → (2 + 3) * 4 → 5 * 4 → 20
❷ result1 = a + (b * c); → 2 + (3 * 4) → 2 + 12 → 14
```

1 Arithmetic Operators

데이터 형식 변환

- **데이터 형식의 강제 형 변환**
  - 가정: int a = 2, b = 3, c = 4;
    double results;

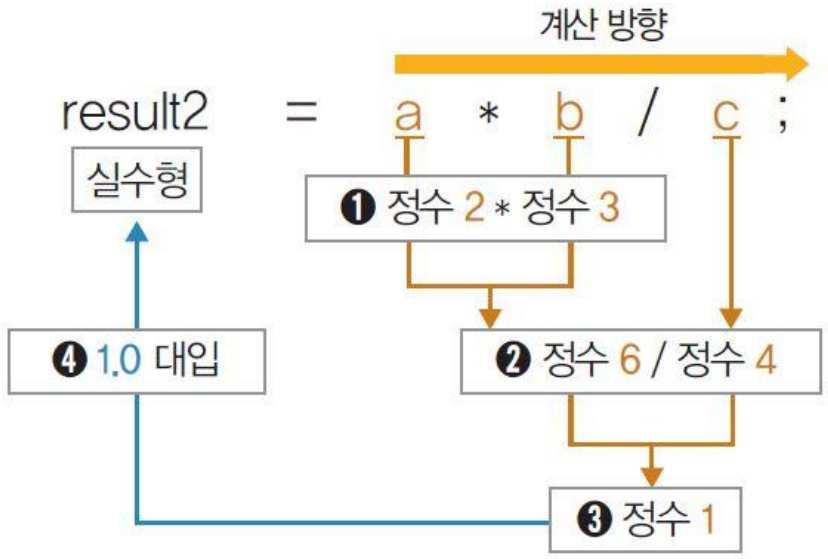


그림 4-1 강제 형 변환을 하지 않았을 때의 결과

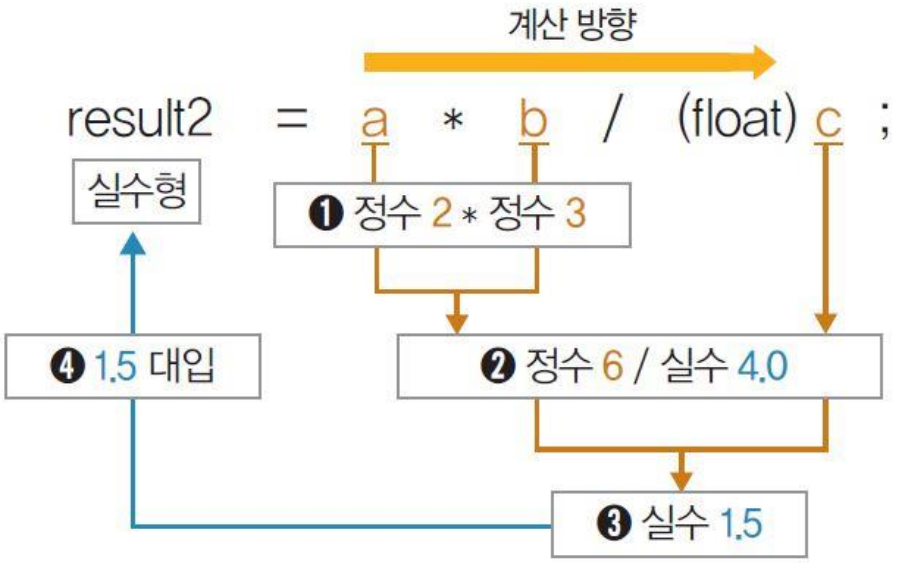


그림 4-2 강제 형 변환을 했을 때의 결과

- **강력한 기능을 갖는다**
  - 즉, 친절한 언어가 아니다.
  - 이런 변환 과정을 직접 수동으로, 그리고 의도적으로 활용할 수 있다.

1 Arithmetic Operators

증감 연산자

- **증감 연산자: ++, --**
  - 단항 연산자 (Unary Operator)
  - 연산자의 앞에 변수를 쓸 수도, 뒤에 변수를 쓸 수도 있다. (단, 연산 결과가 다름)

연산자	명칭	사용 예	설명
++	증가 연산자	a++ 또는 ++a	a += 1 또는 a = a + 1과 동일하다.
--	감소 연산자	a-- 또는 --a	a -= 1또는 a = a - 1과 동일하다.

- **설명의 추가 설명**
  - 결과는 같으나 과정이 다르다
  - 기억하자.

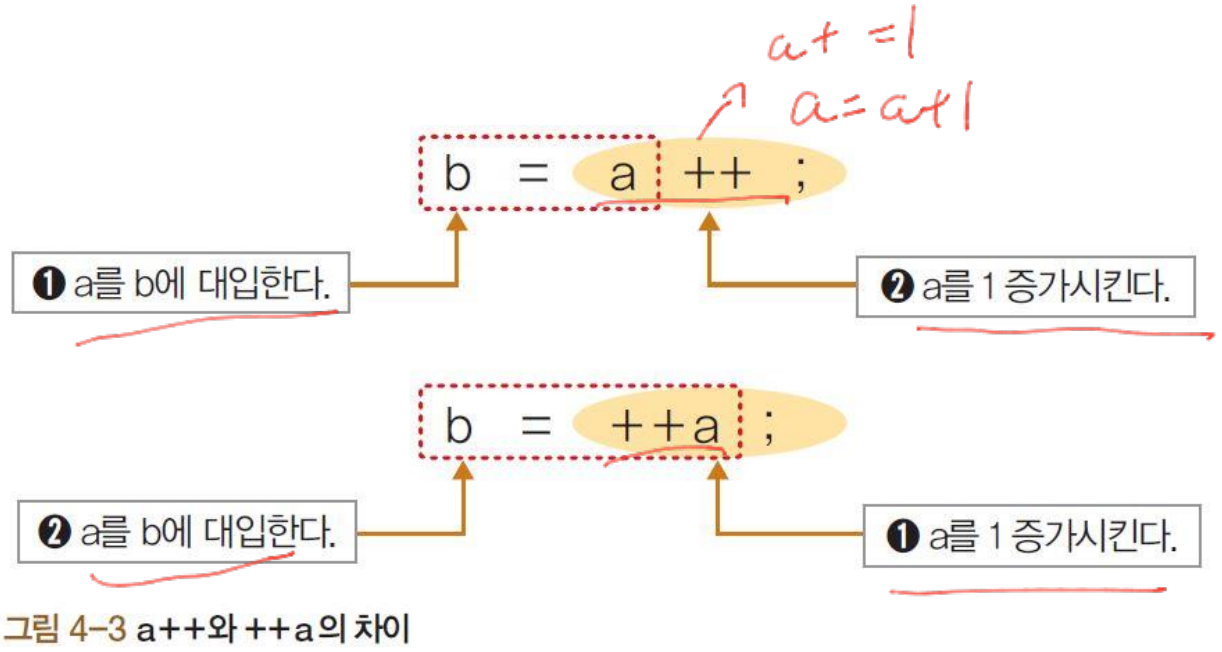


그림 4-3 a++와 ++a의 차이

1 Arithmetic Operators

단축 대입 연산자

- **단축 대입 연산자:** +=, -=, *=, /=, %=
  - 이항 연산자 (Binary Operator)

연산자	명칭	사용 예	설명
+=	대입 연산자	a += 3	a = a + 3과 동일하다.
-=	대입 연산자	a -= 3	a = a - 3과 동일하다.
*=	대입 연산자	a *= 3	a = a * 3과 동일하다.
/=	대입 연산자	a /= 3	a = a / 3과 동일하다.
%=	대입 연산자	a %= 3	a = a % 3과 동일하다.

- **설명의 추가 설명**
  - 결과는 같으나 과정이 다르다
  - 기억하자.

2 Relational Operators

관계 연산자의 기본 개념

- **수치적 관계를 측정**
  - 관계 연산자 (또는 비교 연산자) 는 어떤 것이 큰지, 작은지, 같은지 비교하는 것
  - 결과는 참(True, 1) 이나 거짓 (False, 0)
  - 주로 조건문이나 반복문에 사용, 대체로 단독으로 쓰이지 않음.
  - 단독으로 사용할 수도 있음, 또한 단독으로 사용할 줄도 알아야 함.

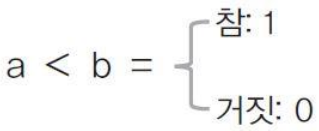


그림 4-4 관계 연산자의 기본 개념

표 4-3 관계 연산자

연산자	의미	설명
==	같다.	두 값이 동일하면 참이다.
!=	같지 않다.	두 값이 다르면 참이다.
〉	크다.	왼쪽이 크면 참이다.
〈	작다.	왼쪽이 작으면 참이다.
〉=	크거나 같다.	왼쪽이 크거나 같으면 참이다.
〈=	작거나 같다.	왼쪽이 작거나 같으면 참이다.

2 Relational Operators

관계 연산자의 기본 개념

- **아래 그림의 동작 결과를 예상해보라.**

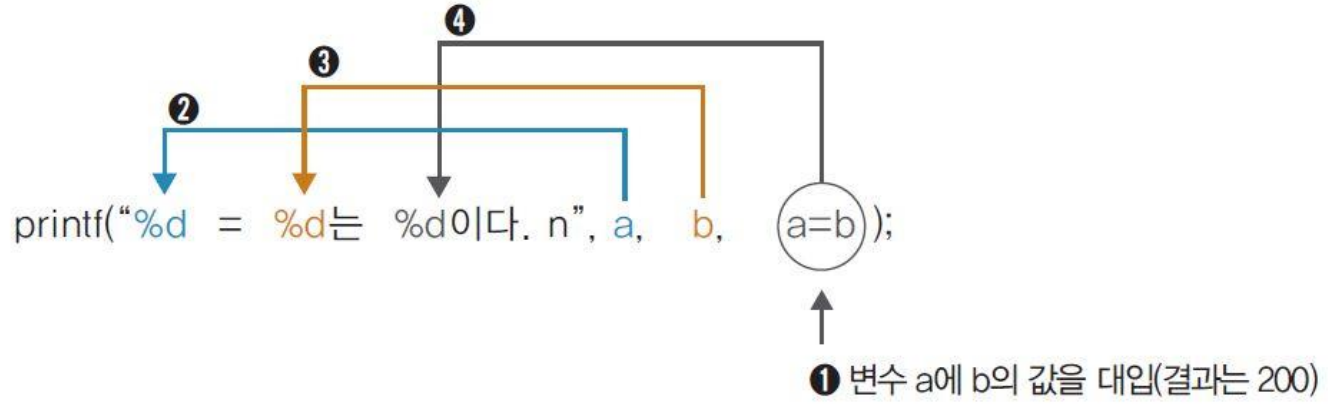


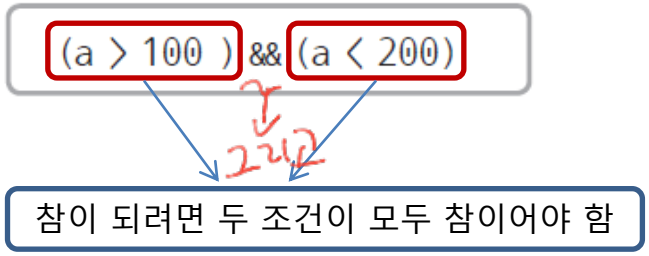
그림 4-5 대입 연산자의 작동

- 의도한 결과: a = b는 참이다. 또는 a = b 는 거짓이다.
- 실제 결과는?

- **이 기능 또한 의도적으로 설계하고 사용된다.**
  - 언제? 파일의 반복적인 입출력에서
  - 당장 사용하기는 어렵더라도 기억하자.
    - ➔ 조건문/반복문의 조건식은 관계/논리만 가능한 것이 아니다.

2 Logical Operators
논리 연산자

- **진리값 (참, 거짓) 사이의 연산**
  - AND, OR, NOT



- **논리 연산자의 종류**

표 4-4 논리 연산자

연산자	의미		사용 예	설명
&&	~이고	그리고(AND)	(a>100) && (a<200)	둘 다 참이어야 참이다.
||	~이거나	또는(OR)	(a>100) || (a<200)	둘 중 하나만 참이어도 참이다.
!	~아니다	부정(NOT)	!(a==100)	참이면 거짓, 거짓이면 참이다.

3 Bit Operators
비트 연산자

- **실제적 비트 연산**
  - 정수로 표현되는 모든 데이터들 사이에 가능한 연산
  - 먼저, 피연산자를 비트로 표현하고
  - 이 둘 사이에 연산을 수행

- **비트 연산자의 종류**

표 4-5 비트 연산자

연산자	명칭	설명
&	비트 논리곱(AND)	둘 다 1이면 1이다.
|	비트 논리합(OR)	둘 중 하나만 1이면 1이다.
^	비트 배타적 논리합(XOR)	둘이 같으면 0, 둘이 다르면 1이다.
~	비트 부정	1은 0으로, 0은 1로 변경한다.
<<	비트 왼쪽 시프트(이동)	비트를 왼쪽으로 시프트(이동)한다.
>>	비트 오른쪽 시프트(이동)	비트를 오른쪽으로 시프트(이동)한다.

3 Bit Operators

논리곱, 논리합 연산자

- **논리곱 연산자와 논리합 연산자: &와 |**
  - 10진수를 2진수로 변환한 후 각 비트에 AND 또는 OR 연산 수행

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

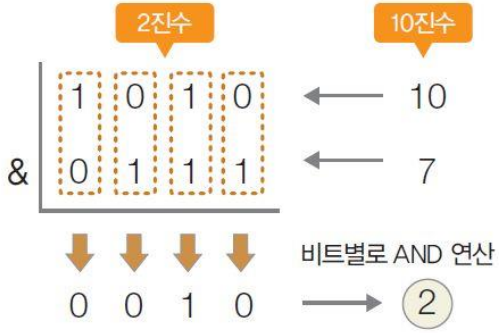


그림 4-6 비트 논리곱의 예

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

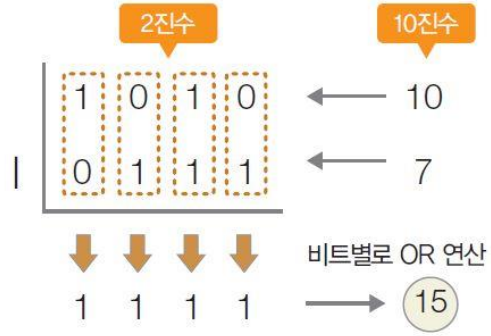


그림 4-7 비트 논리합의 예

- **비트 배타적 논리합 연산자: ^**
  - 두 값이 다르면 참(1), 같으면 거짓(0)

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

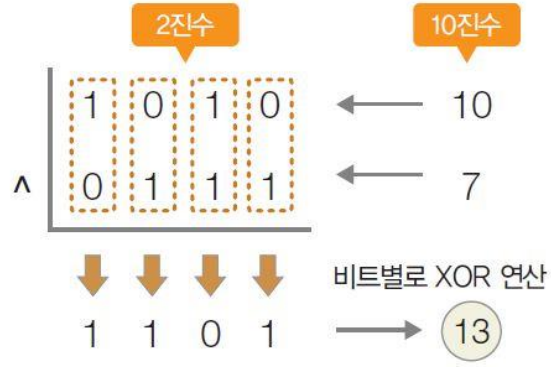


그림 4-8 비트 배타적 논리합의 예

- **결과를 10진수, 2진수, 16진수로 표현해보자**

3 Bit Operators
비트 마스크 연산

- **비트 마스크(mask)를 이용한 비트 곱,합 연산**
  - 특정 비트를 0 또는 1로 변환

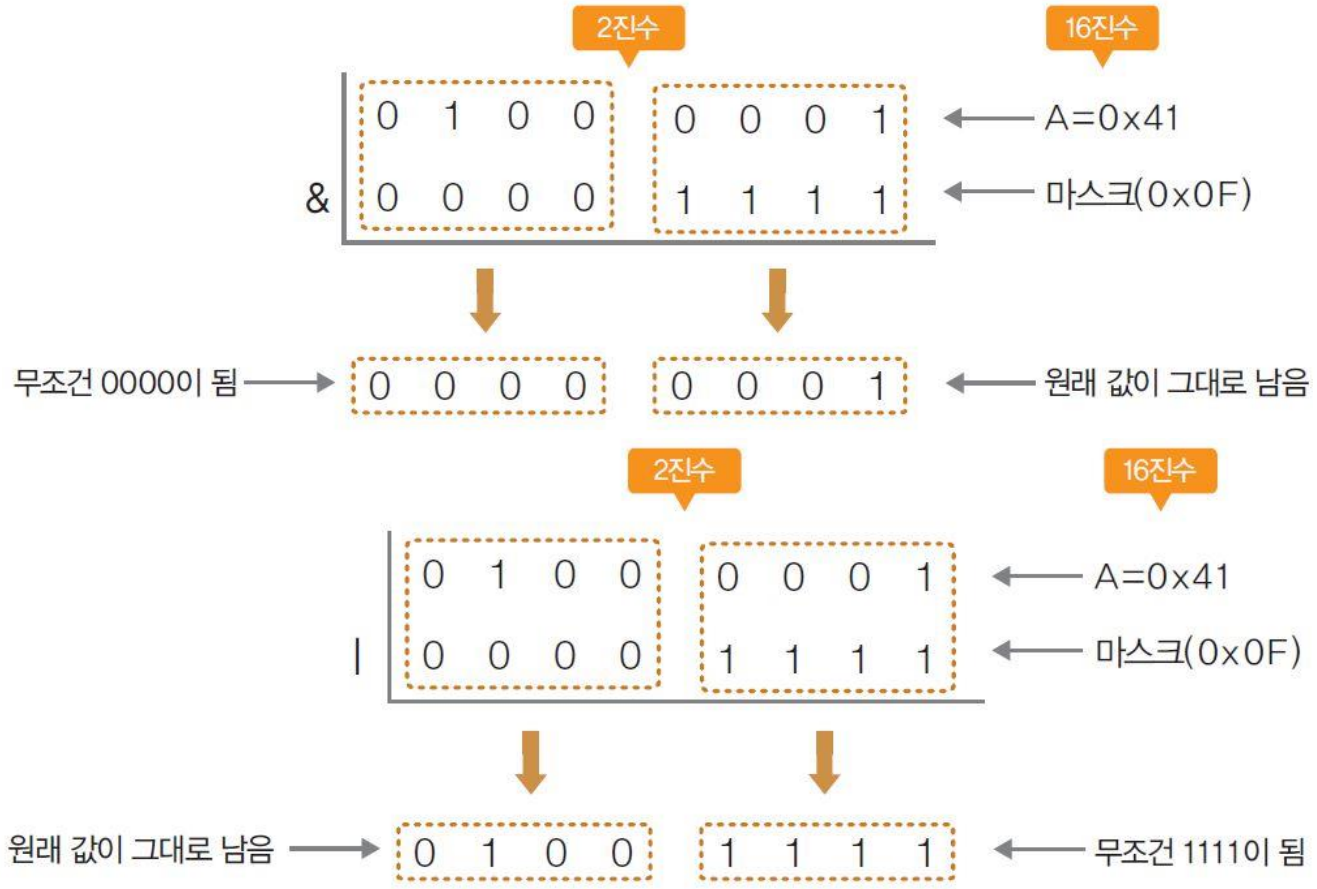


그림 4-10 마스크 0x0F를 사용한 비트 논리합의 예

3 Bit Operators
비트 부정 연산자

- **비트 부정(~) 연산자**
  - 두 수에 대한 연산이 아니라 개별 비트의 값을 반대로 만든다.
  - 어떤 수의 음숫값(-)을 찾을 때 사용

  - cf) 2의 보수(음수) = { 1의 보수(각 비트의 값을 반전시킨 값) } + 1

3 Bit Operators
비트 시프트 연산자

- **비트 왼쪽 시프트(<<) 연산자**
  - 나열된 비트를 왼쪽으로 시프트(shift)하는 연산자
  - 왼쪽 시프트를 할 때마다 2^n(2^1, 2^2, 2^3...)을 곱한 효과

  - 26_{10} 을 왼쪽으로 시프트 연산한 예
    - 0001 1010_2 로 변환한 후 비트 이동

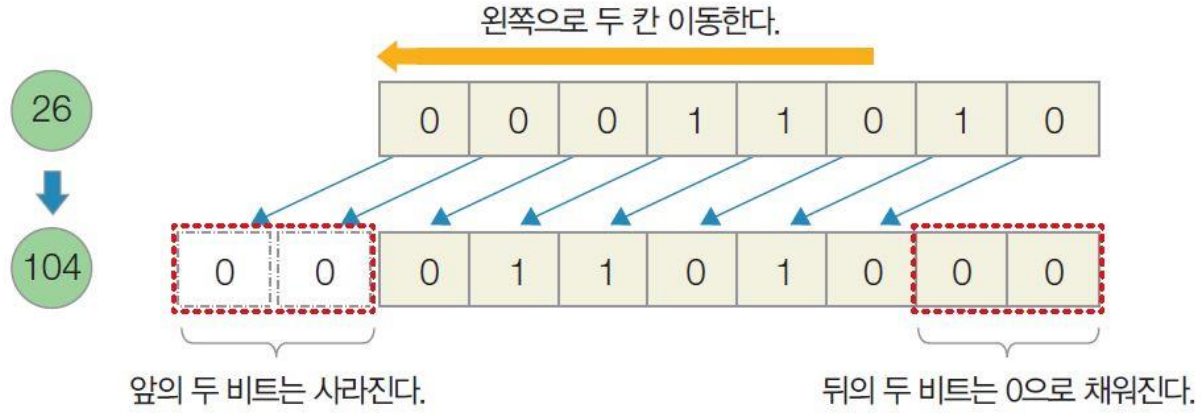


그림 4-11 26을 왼쪽으로 두 칸 시프트한 결과

3 Bit Operators
비트 시프트 연산자

- **비트 오른쪽 시프트(>>) 연산자**
  - 나열된 비트를 오른쪽으로 시프트하는 연산자
  - 오른쪽으로 시프트할 때마다 2^n(2^1, 2^2, 2^3...)으로 나눈 효과

  - 26_{10} 을 오른쪽으로 시프트 연산한 예

그림 4-12 26을 오른쪽으로 두 칸 시프트한 결과

4 Operator Priority

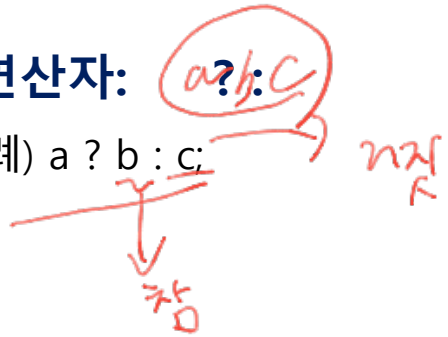
연산자 우선 순위

표 4-6 연산자 우선순위

우선순위	연산자	명칭	순위가 같을 경우 진행 방향
1	( ) [ ] . ->	1차 연산자	→
2	+ - ++ -- ~ ! * &	단항 연산자(변수 또는 상수 앞에 붙음)	←
3	* / %	산술 연산자	→
4	+ -	산술 연산자	→
5	<< >>	비트 시프트 연산자	→
6	<<= >>=	비교 연산자	→
7	== !=	동등 연산자	→
8	&	비트 연산자	→
9	^	비트 연산자	→
10	|	비트 연산자	→
11	&&	논리 연산자	→
12	||	논리 연산자	→
13	?:	삼항 연산자	→
14	= += -= *= /= %= &= ^= |= <<= >>=	대입 연산자	←
15	,	콤마 연산자	→

A Ternary Operators
추가) 삼항 연산자

- **삼항 연산자: ? :**
  - 용례) a ? b : c;



  - a에는 진리값을 반환하는 조건식
  - 참일 경우 b를 거짓일 경우 c를 수행

  - 코드 가독성과 간결성을 높여주는 보편적인 연산자

- **마치 if-else 처럼 동작**