```
System.out.println(1 + x << 33);
```

'1 + x << 33'는 x의 값이 2이므로 '1 + 2 << 33'가 된다. 덧셈연산자(+)보다 쉬프트연산자(<<)가 우선순위가 낮으므로 '3 << 33'이 된다. int는 32 bit이므로 33번 쉬프트 하지 않고 1번만 쉬프트 한다. '3 << 1'은 3에 '2의 1제곱' 인 2를 곱하는 것과 같은 결과를 얻으므로 '3 * 2'가 되어 결국 6을 얻는다.

```
System.out.println(y += 10 - x++);
```

'y += 10 - x++'를 풀어쓰면, 'y = y + (10 - x++)'이 된다. x++은 후위형이기 때문에 x의 값이 증가되지 않은 상태에서 (10 - x)는 계산되고 x의 값은 1증가된다.
그래서 (10 - 2)로 계산이 되고 x의 값은 1증가하여 3이 된다. y의 값은 5이므로 식은 'y = 5 + (10 - 2)'가 되어 y에 13이 저장된다.

```
System.out.println(c+1);
```

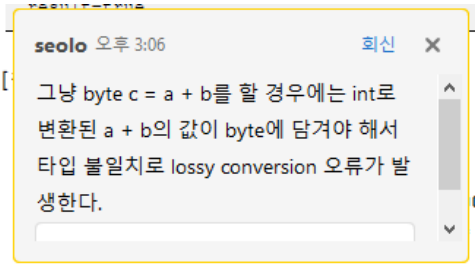
c+1은 c의 값이 'A'이므로 'A'+1이 되고, 이항연산자의 성질(int보다 작은 타입은 int로 변환후 연산)때문에 'A'는 문자코드 값인 65로 변환되어 '65 + 1'을 수행하여 66을 결과로 얻는다. 단지 변수 c에 저장된 값을 읽어서 수식을 계산한 것이므로 변수 c의 저장된 값에는 아무런 변화가 없다.

```
System.out.println(++c);
```

단항연산자인'++'은 이항연산자와 달리 int보다 작은 타입도 형변환을 하지 않는다.(이항연산자는 연산을 위해 '피연산자 스택(operand stack)' 을 사용하는데 이 과정에서 형변환이 발생하는 것이다. 반면에 단항연산자인 증가연산자'++'은 '피연산자 스택' 을 사용하지 않으므로 형변환도 발생하지 않는다.) 그래서 println은 변수 c를 숫자(int)로 출력하는 것이 아니라 문자로 출력한다. 변수 c에 저장된 문자가 'A'(실제로 저장된 것은 'A'의 문자코드인 65)이므로 문자코드의 값이 1증가되어 66('B'의 문자코드)이 변수 c에 저장된다.
변수 c에 저장된 것은 문자코드, 즉 정수값이다. println은 이 값을 타입에 따라 어떻게 출력할지를 결정한다. 만일 문자타입이면 저장된 값(문자코드)에 해당하는 문자를 출력하고 숫자라면 숫자로 출력한다.

```
System.out.println(c++);
```

단항연산자'++'이 후위형인 경우에는 println()에 의해서 변수 c가 출력된 후에 c에 저장된 값이 증가하므로 문자 'B'가 출력된 후에 변수 c의 값이 1증가해서 문자 'C'가 저장된다.




```
0 * 3000 * 3000L;
(float)d==f2;
```

[해설] 이항연산은 두 피연산자의 타입을 일치시킨 후 연산을 수행한다는 것,그리고 int보다 작은 타입은 int로 자동변환한다는 것은 반드시 기억하고 있어야하는 중요한 내용이다.

```
byte c = a + b;  →  byte c = (byte)(a + b);
```

변수 a와 b는 모두 byte타입이므로 이항연산인 덧셈연산을 하기 전에 int타입으로 자동형변환된다. int와 int의 덧셈이므로 연산결과는 int가 된다. int타입의 값(4 byte)을 byte타입의 변수(1 byte)에 담아야하므로 형변환을 해주어야한다.

```
ch = ch + 2;    →  ch = (char)(ch + 2);
```

이것도 이전의 경우와 마찬가지이다. char타입이 덧셈연산의 과정을 거치면서 int타입으로 변환되므로 char타입의 변수에 저장하기 위해서는 char타입으로 형변환을 해주어야한다.

```
long l = 3000 * 3000 * 3000;  →  long l = 3000 * 3000 * 3000L;
```

int*int*int의 결과는 int이므로 int타입의 최대값인 약 2*10의 9제곱을 넘는 결과는 오버플로우가 발생하여 예상한 것과는 다른 값을 얻는다. 그래서 피연산자 중 적어도 한 값은 long타입이어야 최종결과를 long타입의 값으로 얻기 때문에 long타입의 접미사 'L'을 붙여서 long타입의 리터럴로 만들어줘야 한다.

```
boolean result = d==f2;  →  boolean result = (float)d==f2;
```

비교연산자도 이항연산자이므로 연산 시에 두 피연산자의 타입을 맞추기 위해 형변환이 발생한다. 그래서 double과 float의 연산은 double과 double의 연산으로 자동형변환 되는데, 실수는 정수와 달리 근사값으로 표현을 하기 때문에 float를 double로 형변환했을 때 오차가 발생할 수 있다.
그래서 float값을 double로 형변환하기 보다는 double값을 유효자리수가 적은 float로 형변환해서 비교하는 것이 정확한 결과를 얻는다.

[3-10] 다음은 대문자를 소문자로 변경하는 코드인데, 문자 ch에 저장된 문자가 대문자인 경우에만 소문자로 변경한다. 문자코드는 소문자가 대문자보다 32만큼 더 크다. 예를 들어 'A'의 코드는 65이고 'a'의 코드는 97이다. (1)~(2)에 알맞은 코드를 넣으시오.

[연습문제]/ch3/Exercise3_10.java

```
class Exercise3_10 {
  public static void main(String[] args) {
      char ch = 'A';

      char lowerCase = ('A' <= ch && ch <= 'Z') ? (char)(ch+32) : ch;

      System.out.println("ch:"+ch);
      System.out.println("ch to lowerCase:"+lowerCase);
  }
}
```

[실행결과]

```
ch:A
ch to lowerCase:a
```

[정답] ('A' <= ch && ch <= 'Z'), (char)(ch+32)
[해설] 대문자인 경우에만 문자코드의 값을 32만큼 증가시키면 소문자가 된다. 문자 ch가 대문자인지를 확인하는 조건식은 'A' <= ch && ch <= 'Z'이고, 문자 ch의 문자코드를 32증가시키기 위해서는 덧셈연산을 해야하는데, 이 때 덧셈연산의 결과가 int이므로 char타입으로의 형변환이 필요하다.

● 이항 연산자는 피연산자의 타입이 int보다 작을 경우(byte, char, short) int형으로 형변환 후 연산을 수행하기 때문에, 연산 수행 결과를 그대로 출력하면 int형으로 출력된다. 만약 연산 수행 결과를 다시 피연산자 변수에 저장하고 싶을 경우, 해당 타입(byte, char, short)으로 다시 캐스팅해주어야 한다.

Resource: 자바의 정석