# 4. 연산자

| 구 분 | | 연 산 자 | 결합순서 | 우선순위 |
|---|---|---|---|---|
| 일차연산자 | | ( ), [ ], . , –> | → | 높다 |
| 단항연산자 | | –, + +, – –, ~, !, *, &, sizeof, (형) | ← | ↑ |
| 이항연산자 | 산술연산자 | *, /, % | → | |
| | 산술연산자 | +, – | | |
| | 비트이동 | 〉〉, 〈〈 | | |
| | 대소비교 | 〉, 〉=, 〈, 〈= | | |
| | 등가비교 | ==, != | | |
| | 비트 AND | & | | |
| | 비트 XOR | ∧ | | |
| | 비트 OR | \| | | |
| | 논리 AND | && | | |
| | 논리 OR | ‖ | | |
| 조건연산자 | | ? : | ← | |
| 대입연산자 | | =, + =, – =, * =, / =, % =<br>〉〉=, 〈〈=, & =, ∧ =, \| = | ← | ↓ |
| 나열연산자 | | , | → | 낮다 |

• C에는 지수연산자가 없다. **지수연산**은 함수를 사용한다. **[y = pow(4, 2) = 16]**

◉ **연산자 결합순서**

① 결합순서는 연산우선순위가 같은 연산자가 수식 내에서 공통의 피연산자를 공유할 때 어느 방향으로 적용할 것인가에 대한 결합 규칙이다.

② 즉, 결합 규칙은 피연산자를 공유하는 우선순위가 같은 연산자에 적용되는 규칙이다.

**[예제] 수식 '24 / 3 ＊ 2'에 대한 결합순서**

| | |
|---|---|
| **왼쪽에서 오른쪽으로 적용(→)** | 24 / 3 * 2 = ((24 / 3) * 2) = 8 * 2 = **16** |
| **오른쪽에서 왼쪽으로 적용(←)** | 24 / 3 * 2 = (24 / (3 * 2)) = 24 / 6 = **4** |

↓

↓ **분석**

↓

㉠ 피연산자 3은 연산 우선순위가 같은 나눗셈(/)과 곱셈(*) 연산자가 공유한다.

㉡ 나눗셈(/)과 곱셈(*) 연산자는 결합순서가 왼쪽에서 오른쪽이다.(→)

㉢ 따라서, 수식은 ((24 / 3) * 2) = 8 * 2 = **16**이 된다.

### (1) 대입연산자 / 비교연산자

**대입**연산자 기호 =을 사용하고, **비교**연산자 기호 ==을 사용한다.

| | | |
|---|---|---|
| **대입 연산자** | a = 2;<br>a = a + 3;<br>a += 3;<br>a *= 4; | a에 2를 대입시킨다.<br>a의 값을 3만큼 증가시킨다.<br>a의 값을 3만큼 증가시킨다. (a = a + 3)<br>a의 값을 4배 증가시킨다. (a = a * 4) |
| **비교 연산자** | a = 2;<br>**if(a == 2)** | 연산자 ==은 a의 값이 2인지를 비교하여 참 또는 거짓을 구한다.<br>a=2이므로, **비교 결과**는 같으므로 **참(1)**이 된다. |
| | a = 2;<br>b = 3;<br>**y = a == b;** | a와 b를 비교한 결과, 값이 서로 다르므로 **거짓(0)**이 된다. |

- 좌우를 비교할 때는 연산자 '=='을 사용한다. 연산자 '='을 사용하는 것이 아니다.**(주의!)**
- C에서 **참은 1, 거짓은 0**이 된다.

### (2) 관계연산자(relational operator)

관계연산자가 사용된 식이 성립하면(참이면) 1, 성립하지 않으면(거짓이면) 0의 값을 갖는다.
즉, C에서 **참, 거짓**은 궁극적으로 **1과 0**으로 처리한다.

| | |
|---|---|
| **a = 6;** | |
| **b = a > 2;** | b에는 1이 대입된다.**(참)** |
| **c = a < 2;** | c에는 0이 대입된다.**(거짓)** |

**// C의 조건문에서 참, 거짓**

**if(0) → 거짓**
**if(1) → 참**
**if(2) → 참**
**if(1.5) → 참**

⇒ 즉, C의 조건문에서는 **0은 거짓, 0 이외의 모든 값은 참**으로 처리된다.

### (3) 연산자 %

연산자 %는 정수 연산 결과에서 나머지를 구한다.(**modulo** 연산)

| | | | |
|---|---|---|---|
| [예] | 5 % 2 = 1 | -5 % 2 = -1 | -2 % 5 = -2 |
| | 2 % 5 = 2 | 5 % -2 = 1 | 2 % -5 = 2 |
| | | -5 % -2 = -1 | -2 % -5 = -2 |

- 연산자 %의 피연산자에 **음수**가 있으면 결과는 **앞의 피연산자 부호**를 가진다.

### (4) 단항연산자(unary operator)

한 개의 피연산자를 대상으로 연산을 수행한다.

| | |
|---|---|
| **전위식** | ++a; → **a의 현재 값이 1 증가된 후 수식 중에 사용된다.** |
| **후위식** | a++; → **a의 현재 값이 수식 중에 사용된 후 1 증가된다.** |

• 단항연산자 ++, --가 수식 중에 사용되면 위치에 따라 결과가 다르게 된다.

**예제 1**

**• 전위식**

```
a = 3;
b = 2 * ++a;
```

**결과 : a = 4, b = 8**

**• 후위식**

```
a = 3;
b = 2 * a++;
```

**결과 : a = 4, b = 6**

**예제 2**

**a = 5, b = 70;일 때 수식 y=a+++b;가 수행된 후의 y, a, b의 값은?**

**(풀이)** 먼저, 수식 y=a+++b;의 코딩 형태를 잘 살펴보면, 피연산자 사이에 연산자가 많이 사용되면서 띄어쓰기가 전혀 되지 않았다.
따라서, 컴파일러는 주어진 수식을 다음 2가지 형태로 분석할 수 있다.

**• 분석 1 (정답)**

```
y = a++ + b;
```

**값 : y = 75, a = 6, b = 70**

**• 분석 2 (오답)**

```
y = a + ++b;
```

~~**값 : y = 76, a = 5, b = 71**~~

**(설명)** 단항연산자 ++의 **결합순서는 우에서 좌**(←)이다.
따라서, 컴파일러는 '**분석 1**'처럼 주어진 수식을 처리한다.

[Tip] 수식이 "**y = a++ + b;**"나 "**y = a + ++b;**"처럼 원시코드에서 **띄어쓰기**가 되어 있으면 주어진 **띄어쓰기** 형태 그대로 수행된다. 즉, 띄어쓰기가 된 형태로 수행된다.

### (5) 비트연산자(bitwise operator)

자료 표현의 최소 단위인 비트를 직접 처리하는 연산자이다.

| 구 분 | 연산자 | 기 능 | 사용 예 |
|---|---|---|---|
| 이동연산자 | 〈〈<br>〉〉 | 비트 값을 좌측으로 이동<br>비트 값을 우측으로 이동 | a = 16;<br>r = a 〈〈 3 = 16 * $2^3$ = 16 * 8 = **128**<br>r = a 〉〉 3 = 16 / $2^3$ = 16 / 8 = **2** |
| 논리연산자 | &<br>\|<br>∧<br>~ | 비트 논리곱(AND)<br>비트 논리합(OR)<br>비트 배타적 논리합(XOR)<br>반전 ( NOT, 1의보수 ) | r = a & b;<br>r = a \| b;<br>r = a ∧ b;<br>r = ~a; |

**[예] 자료가 2진수로 8비트에 저장되어 있는 경우**

```
    0011 1111        0011 1111        0011 1111
& ) 0111 0011    | ) 0111 0011    ∧ ) 0111 0011
    0011 0011        0111 1111        0100 1100
```

### (6) 조건연산자 [ ? : ]

조건연산자(conditional operator)는 피연산자가 **3개의 항**으로 구성되어 있다.
일명 **3항연산자**라고 한다.

---

**[형식]** 조건 ? 표현1 : 표현2 ; → 조건이 **참이면 표현1**, **거짓이면 표현2**가 수행된다.

---

| | | |
|---|---|---|
| [예1] | a = 8, n = 5;<br>(a > 9) ? n++ : n--; | • **조건** 'a 〉 9'가 **거짓**이므로 **n--**이 수행된다.<br>• 즉, n의 값은 1 감소하여 4가 된다. |
| [예2] | a = 8, b = 7;<br>y = (a > b) ? a+b : a-b; | • 먼저, 조건에서 a가 b보다 크므로 조건은 참<br>• 'a+b'가 수행되고, 그 결과 15가 y에 대입된다. |
| [예3] | a = 8, n = 5;<br>y = (a > 9) ? n++ : n--; | • **조건** 'a 〉 9'가 **거짓**이므로 **n--**이 수행된다.<br>• 주의할 것은 단항연산자가 변수 뒤에 있으므로(**n--**)<br>• n값이 y에 대입된 후에 n의 값이 1감소된다.<br>• 실행 후 y = 5, n = 4가 된다. |

### (7) 나열연산자[ , ]

나열연산자는 일명 콤마연산자라고 한다.

수식을 콤마로 구분하여 나열하고 연산은 왼쪽부터 오른쪽으로 차례로 진행된다.

| | | |
|---|---|---|
| [예1] | a = **1**, 2, 3, 4, 5, 6, 7, 8, 9;<br>a = (1, 2, 3, 4, 5, 6, 7, 8, **9**); | a=1 (연산자 =이 나열연산자 ,보다 빠르다)<br>a=9 (괄호부터 연산되므로 a=9가 된다) |
| [예2] | a = b = 1, 2, 3;<br>a = b = (1, 2, 3);<br>a = (b = 1, 2, 3); | a=1, b=1(2와 3은 아무런 의미가 없다)<br>a=3, b=1(1과 2는 아무런 의미가 없다)<br>a=3, b=1(b에 1이 대입되고, a에 3이 대입된다) |
| [예3] | r = ( a = 5, b = a + 2, 3 * b); | • 먼저, a에 5가 대입되고,<br>• a + 2의 결과인 7이 b에 대입되고,<br>• 3 * b의 결과인 21이 r에 대입된다. |

### (8) 형변환연산자[(자료형)]

형변환연산자(cast operator)는 자료 값은 그대로 두고 **자료형을 강제적으로 바꾼다.**

예를들면, 정수형을 실수형으로 자료형을 명시적(explicit)으로 바꿀 때 쓰인다.

| | | |
|---|---|---|
| [예1] | **int a = 5;**<br>**double b = 3.9, c;**<br>**c = (double)a + b;** | 정수형 a를 실수형(**double**형)으로 바꾸어 연산을 수행한다.<br>이때 사용된 **(double)**가 **형변환연산자**로 사용된 것이다.<br>**c** = (double)a + b = 5.0 + 3.9 = **8.9** |
| [예2] | **int k1, k2;**<br>**k1 = 1.7 + 1.8;**<br>**k2 = (int)1.7 + (int)1.8;** | 연산 결과 **k1** = 1.7 + 1.8 = 3.5 = **3**<br>연산 결과 **k2** = 1 + 1 = **2** |

### (9) 주소연산자[&]

주소연산자(address operator)는 어떤 변수에 해당하는 **기억장소의 주소**를 구한다.

즉, &a는 변수 a가 메모리 상에 위치하는 기억장소의 시작주소이다.

[예] long a;

p = &a ; → 변수 a가 위치하는 기억장소의 시작 주소가 p에 대입된다.

(이 때 변수 p는 **포인터** 변수이어야 한다)

[Tip] 포인터 변수는 기억공간의 주소를 보관할 수 있는 자료형이다.(뒤에서 다룬다)

### (10) sizeof 연산자

자료형, 변수, 수식의 결과 등이 차지하는 기억공간의 **바이트 수**를 구한다.

| | |
|---|---|
| sizeof(char) | 자료형 'char'의 크기 1을 구한다. |
| sizeof(size) | 'size'가 차지하는 기억공간의 바이트 수를 구한다. |

**[예] 16비트 컴퓨터 시스템에서**

```
char ch = 65;
long a, b;
b = sizeof(ch) + sizeof(long); → b의 값은 1 + 4 = 5이다.
```

◈ sizeof 연산자(16비트와 32비트 컴퓨터 시스템에서 비교)

| 사용 예 | 16비트 | 32비트 | 설 명 |
|---|---|---|---|
| char a;<br>y = sizeof(a); | 1 | 1 | |
| int b;<br>y = sizeof(b); | 2 | 4 | int는 cpu 레지스터 크기이다(word 크기)<br>16 또는 32비트 컴퓨터는 각각 16 또는 32비트 |
| long c;<br>y = sizeof(c); | 4 | 8 | long은 시스템에 따라 4byte로 취급하기도 한다.<br>예 : OS windows는 int와 long 둘 다 4byte 취급 |
| char *a;<br>y = sizeof(a); | 2 | 4 | 포인터 크기를 구한다. 포인터 형과는 무관하다.<br>포인터 크기는 int형 크기와 같다.<br>포인터 변수도 cpu의 레지스터를 그대로 이용하기 때문에 int 크기가 포인터 크기가 된다. |
| int *b;<br>y = sizeof(b); | 2 | 4 | |
| long *c;<br>y = sizeof(c); | 2 | 4 | |
| char *a;<br>y = sizeof(*a); | 1 | 1 | 포인터에서 *가 붙어 있으면 각 포인터형 크기만큼 자료를 가져온다.<br><br>long은 시스템에 따라 4byte로 취급하기도 한다. |
| int *b;<br>y = sizeof(*b); | 2 | 4 | |
| long *c;<br>y = sizeof(*c); | 4 | 8 | |
| char a[5] = "KS";<br>y = sizeof(a); | 5 | 5 | 배열의 크기를 구한다.<br>만약, char a[ ] = "KS"이면 크기는 3 ('\0' 포함) |
| char *p = "D";<br>y = sizeof(p); | 2 | 4 | 포인터 변수 크기를 구한다. |
| char a = 3;<br>y = sizeof(2*a); | 2 | 4 | 정수 연산의 최소 단위는 int 형이므로 |

- 배열과 포인터 부분은 뒤에서 설명한 배열과 포인터를 이해하고, 살펴보면 될 것이다.
- 자료형 크기는 OS(16비트, 32비트, 64비트)나 컴파일러에 따라 조금씩 다르다.
- 특히, long은 4byte로 취급하는 OS가 많다.(OS windows : int와 long 둘 다 4byte로 취급)

## 기출문제 분석

### 1. 〈보기〉 C 프로그램의 출력은? [2018년 서울 9급]

```
------<보기>--------------
 #include <stdio.h>
 int main()
 {
   int a = 5, b = 5;
   a *= 3 + b++;
   printf("%d %d", a, b);
   return 0;
 }
--------------------------
```

① 40 5　　② 40 6
③ 45 5　　④ 45 6

♤ C 프로그램

---

• a *= 3 + b++;
　　↓
a = a * (3 + b++); //괄호로 묶는 것에 주의할 것
　　↓
a = 5 * (3 + 5) = 5 * 8 = **40**
b++이므로, b = b + 1 = 5 + 1 = **6**　　//b는 나중에 1 증가한다.

---

정답 : ②