

Expressions

소프트웨어학부
박영훈 교수

이 단원의 목표

- 산술연산자의 사용법에 대하여 알아본다.

$+$ $-$ $*$ $/$ \div

- 단순 대입 연산자와 복합 대입 연산자의 사용법 대하여 알아본다.
- 증가/감소 연산자의 사용법에 대하여 알아본다.

C에서 사용되는 연산자

- C에서 정의된 연산자는 다음과 같이 분류할 수 있다.

- 산술 연산자

- $+$, $-$, $*$, $/$, $\%$ *→ 나머지*
- (앞에 붙이는) $+$, (앞에 붙이는) $-$

- 관계 연산자 *if문*

- $==$, $!=$, $>$, $<$, $>=$, $<=$

- 논리 연산자

- $\&\&$, $\|\|$, (앞에 붙이는) $!$

- 이진 연산자

- $\&$, $\|$, \wedge , (앞에 붙이는) \sim

마지막 연산자

- 대입 연산자

- $=$, $+=$, $-=$, $*=$, $/=$, $\%=$, $\&=$, $\|=$, $\wedge=$

- 증감 연산자

- (앞에 붙이는) $++$, (앞에 붙이는) $--$
- (뒤에 붙이는) $++$, (뒤에 붙이는) $--$

산술 연산자 (aritmetic)

- C에는 5개의 binary operator (양 옆에 두 개의 operand가 필요한 것) 와 2개의 unary operator (오른쪽에만 operand가 필요한 것) 가 제공된다.

- Binary Operator

$A + B$	$A - B$	$A * B$	A / B	$A \% B$
A와 B의 합	A에서 B를 뺀 값	A와 B의 곱	A에서 B를 나눈 몫	A에서 B를 나눈 나머지

$$10 / 3 = 1$$

- Unary Operator

$+ A$	$- A$
A의 부호를 그대로 유지 (사실상 의미가 없음)	A의 부호를 반대로 바꿈

산술 연산자 /와 %에 대하여 (정수끼리, 실수끼리)

• A / B

- A와 B의 타입이 둘 다 정수일 때, A / B가 갖는 값은 |A|를 |B|로 나눈 몫에 A와 B의 부호가 같으면 +, 다르면 -를 붙인 것이 된다.

- 예:

$$\underline{13 / 5} = \underline{-13 / -5} = 2, \quad \underline{13 / -5} = \underline{-13 / 5} = -2$$

- 만일 A와 B의 타입이 둘 다 실수이면, A / B가 갖는 값은 A를 B로 나눈 실숫값이다.

- 예:

float or double type

$$13.2f / 5.5f = -13.2f / -5.5f = 2.4f,$$
$$13.2f / -5.5f = -13.2f / 5.5f = -2.4f$$

• A % B

- A와 B의 타입이 둘 다 정수일 때만 사용 가능하다. A % B가 갖는 값은 |A|를 |B|로 나눈 나머지에 A의 부호를 붙인 것이 된다.

- 예:

$$\underline{13 \% 5} = \underline{13 \% -5} = 3, \quad \underline{-13 \% 5} = \underline{-13 \% -5} = -3$$

- 공통 사항: B는 0이 될 수 없다.

*$A = B * Q + R$*

연산자의 우선순위 ✓

- 연산자의 우선순위는 높은 순서대로 다음과 같다.

연산자	우선순위	같은 우선순위의 연산자가 둘 이상 있을 때
$+, -$ (Unary)	가장 높음	오른쪽부터 연산
$*, /, \%$	보통	왼쪽부터 연산
$+, -$	가장 낮음	왼쪽부터 연산

- 예:

$i + j * k$ 는 $i + (j * k)$ 와 같다

$- i * - j$ 는 $(- i) * (- j)$ 와 같다

$+ i + j / k$ 는 $(+ i) + (j / k)$ 와 같다.

$- + i$ 는 $-(+ i)$ 와 같다.

☆ 주의

$$a * b / c = (a * b) / c$$

무조건...

대입 연산자 (Assignment Operator)

- 단순 대입 연산자 (Simple Assignment)
 - 변수에 값을 저장만 하는 연산자
 - = 뿐만 존재한다.
- 복합 대입 연산자 (Compound Assignment)
 - 이미 값이 저장되어 있는 변수를 업데이트하는 연산자
 - +=, -=, *=, /=, %= 등이 존재한다.

단순 대입 연산자 (Simple Assignment Operator)

- =의 왼쪽에는 값이 저장될 변수가, 오른쪽에는 상수, 변수, 수식 등이 올 수 있다.

- 예:

`i = 5;` (`i`에 5가 저장됨)

`j = i;` (`j`에 5가 저장됨)

`k = 10 * i + j;` (`k`에 55가 저장됨)

- 만일 =의 왼쪽과 오른쪽의 타입이 다를 경우, 오른쪽의 값이 왼쪽 변수의 타입으로 바뀐다

- 예:

`int i;`

`float x;`

`i = 72.99f;` `// i에 72가 저장됨`

`f = 136;` `// f에 136.0f가 저장됨`

"C는 개발자를 만든다"
에 대한 표지는 X

- 하지만, =의 좌우변의 타입이 다를 경우 어떤 결과가 나올 지 모르는 경우가 있고, 가끔 원하지 않은 결과가 나올 수도 있으므로 웬만하면 좌우변의 타입을 같게 하는 것이 좋다.

=의 부가 기능

- =이 포함된 수식은 그 자체도 값을 갖고 있다.
- 즉, 대입 연산자는 값을 대입하는 기능 뿐 아니라, 그 대입되는 값 자체를 나타내는 부가기능도 제공한다.
- 예를 들어, `i = 3;` 이라 하면, `i`에 3이 저장됨과 동시에, `i = 3;`이라는 수식이 표현하는 값 역시 3이 된다. 즉,

```
printf("%d", i = 3);
```

- 을 실행했을 때 출력되는 값은 3이 된다.
- 이 기능을 잘 활용하면 소스코드를 좀더 간단하게 만들 수 있다.
- 예:

```
i = 0;
```

```
j = 0;
```

```
k = 0;
```

은 `i = j = k = 0;`으로 간단히 할 수 있다. (`i = (j = (k = 0))` 이기 때문)

```
j = i * 2 - 4;
```

```
k = (j + 5) / 3;
```

은 `k = ((j = i * 2 - 4) + 5) / 3;`으로 간단히 할 수 있다.

=는 우선순위가 실행됨



= 의 부가 기능

- =의 부가기능을 사용하고자 할 때, 타입이 다른 경우는 유의해야 한다. 예를 들어,

```
int i;  
float x;  
x = i = 12.34f;
```

라고 하면, i에 정수 12가 저장되고, i = 12.34f가 갖는 값이 정수 12이므로 x에는 실수 12.0이 저장되게 된다.

$\therefore a = (i = 12.34f; i)$

- 부가 기능 사용 예

```
int i, j, k;  
i = 1;  
k = 1 + (j = i);  
printf("%d %d %d", i, j, k);
```

를 실행하면 출력 결과가 어떻게 되겠는가? $\Rightarrow 1 \ 1 \ 2$

Lvalue

Left의 L (대입연산자의 왼쪽에 있을 수 있는 것)

⊕ = 0

- Lvalue란 값이 저장될 수 있는 메모리 공간이다.
- 대부분 변수는 Lvalue가 될 수 있지만, 상수나 수식은 Lvalue가 될 수 없다.
즉, 10이나 2 * i는 Lvalue가 아니다.
- Lvalue가 아닌 operand는 대입연산자의 왼쪽에 있을 수 없다.

예:

```
i = 12;           // OK
12 = i;           // Wrong
i + j = 1;        // Wrong
-i = j;           // Wrong
```

(다: i = -j (0))

- Lvalue가 아닌 operand가 대입 연산자의 왼쪽에 있으면 컴파일 에러가 난다.

복합 대입 연산자 (Compound Assignment Operator)

- 변수 **i**의 값을 업데이트 하고 싶을 때, 예를 들어 **2**를 증가시키고 싶으면 다음과 같이 하면 된다.

`i = i + 2;`

- 이는 복합 대입 연산자를 이용하여 다음과 같이 표현할 수 있다.

`i += 2;`

- `+=` 이외에도 `-=`, `*=`, `/=`, `%=`도 비슷한 기능을 한다. 즉,

`v += e;` 는 **v**를 **e**만큼 증가시킨다.

`v -= e;` 는 **v**를 **e**만큼 감소시킨다.

`v *= e;` 는 **v**에 **e**를 곱한 값을 **v**에 저장한다.

`v /= e;` 는 **v**에서 **e**를 나눈 값을 **v**에 저장한다.

`v %= e;` 는 **v**에서 **e**를 나눈 나머지를 **v**에 저장한다.

복합 대입 연산자의 부가 기능

- 복합 대입 연산자도 =와 마찬가지로 부가 기능이 존재한다.
- 즉, $v += e;$ 는 v 의 값이 e 만큼 증가함과 동시에 $v += e;$ 라는 수식 자체도 그 e 만큼 증가한 값을 갖게 된다.
- 예:

```
int i = 4;  
printf("%d, ", i);  
printf("%d, ", i += 10);  
printf("%d", i);
```

를 실행하면, 맨 처음에는 4가 출력되고, 그 다음에는 $i += 10$ 의 수식이 복합 대입 연산자의 부가 기능에 의해 14라는 값을 가지므로 14가 출력되며, 마지막에는 바로 앞에서 이미 i 값이 14가 되었으므로 역시 14가 출력된다. 결국,

4, 14, 14

가 출력된다.

증가/감소 연산자

- 프로그래밍에서는 어떤 변수를 1만큼 증가시키거나 감소시킬 일이 자주 발생한다 (특히 반복문). 이를 위해서는 다음과 같이 쓸 수 있다.

i = i + 1; i = i - 1;

- 위의 수식을 복합 대입 연산자를 이용해서 나타내면 다음과 같다.

i += 1; i -= 1;

- 위의 식들을 더욱 간단하게 다음과 같이 나타낼 수 있다.

i++; 또는 ++i; i--; 또는 --i;

- 증감연산자의 operand는 반드시 lvalue이어야 한다.

증가/감소 연산자의 부가 기능

- `i++`와 `++i`는 `i`를 1씩 증가시킨다는 공통점이 있지만, 부가 기능은 서로 다르다.

`i++`는 1 증가되기 전의 값을 갖고,
`++i`는 1 증가되고 난 후의 값을 가진다.

- 예를 들어,

`int i = 3;`
`printf("%d", i++);`
`printf(", %d", i);`
=> 출력 결과: 3, 4

전

`int i = 3;`
`printf("%d", ++i);`
`printf(", %d", i);`
=> 출력 결과: 4, 4

후

- `i--`와 `--i`도 마찬가지이다. 즉, `i`를 1씩 감소시키는 것은 같지만,

`i--`는 1 감소되기 전의 값을 갖고,
`--i`는 1 감소되고 난 후의 값을 가진다.



연산자 사용 시 주의사항

- Assignment의 부가 기능을 사용할 때, 같은 변수가 여러 번 등장 하면 예상치 못 한 결과가 나올 수 있다. 예를 들어,

`a = 5;`

`c = (b = a + 2) - (a = 1);`

무엇이 먼저 계산되느냐에 따라

이라고 했을 때, `b = a + 2`가 먼저 계산된다면 c의 값은 6이 될 것이고, `a = 1`이 먼저 계산된다면 c의 값은 2가 될 것이다.

- 이 연산 순서는 컴퓨터마다, 컴파일러마다 다르게 정해지므로, 이런 표현은 피해야 한다.
- 위 연산을 명확하게 쓰려면 다음과 같이 하면 된다. (`b = a + 2`가 먼저 계산되는 것을 기대했을 경우라 가정)

`a = 5;`

`b = a + 2;`

`a = 1;`

`c = b - a;`

- 다음의 예도 마찬가지로 `2 * 2`나 `3 * 2` 둘 다 계산 가능한 시나리오들이다.

`i = 2;`

`j = i * i++;`

연산자 사용 시 주의사항

- C언어에서 명령문으로 모든 수식이 가능. 심지어 대입 연산자가 없는 수식이 명령문으로 존재해도 컴파일 에러가 발생하지 않는다. 즉,

```
int i, j = 3;
```

```
i = 1;
```

```
i++; → i=2
```

```
i * j - 1; : 이자재는 없어요.
```

라고 프로그래밍 해도 전혀 컴파일 에러가 나지 않는다.

- 하지만 `i * j - 1;`는 아무런 의미가 없는 수식으로, 계산 동작만 할 뿐 결과에는 전혀 영향을 미치지 않는다.
- 이러한 실수로 인해 예상치 못한 결과가 나올 수 있으므로 주의해야 한다.