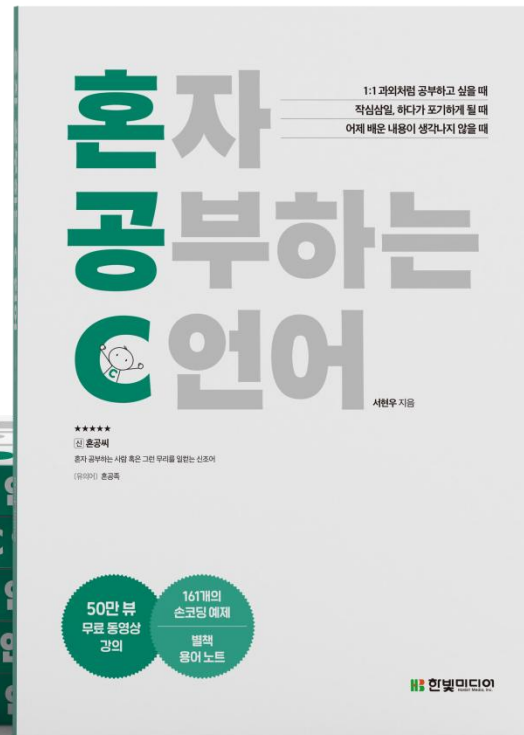


4장 연산자



❖ 산술 연산자와 대입 연산자 (1/2)

대입, 덧셈, 뺄셈, 곱셈, 음수 연산

소스 코드 예제4-1.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a, b;
06     int sum, sub, mul, inv;
07
08     a = 10;                // 대입 연산(=)
09     b = 20;                // 대입 연산(=)
10     sum = a + b;           // 더하기 연산(+) 후 대입 연산(=)
11     sub = a - b;           // 빼기 연산(-) 후 대입 연산(=)
12     mul = a * b;           // 곱하기 연산(*) 후 대입 연산(=)
13     inv = -a;              // 음수 연산(-) 후 대입 연산(=)
14
```

❖ 산술 연산자와 대입 연산자 (2/2)

```
15    printf("a의 값 :%d, b의 값 :%d\n", a, b);
16    printf("덧셈 : %d\n", sum);
17    printf("뺄셈 : %d\n", sub);
18    printf("곱셈 : %d\n", mul);
19    printf("a의 음수 연산 : %d\n", inv);
20
21    return 0;
22 }
```

실행결과

a의 값 :10, b의 값 :20
덧셈 : 30
뺄셈 : -10
곱셈 : 200
a의 음수 연산 : -10

■ 대입 연산자

a = 10;



상수 값을 변수에 저장

sum = a + b;



연산의 결과값을 변수에 저장

❖ 나누기 연산자와 나머지 연산자 (1/2)

- 정수를 나누면 몫이 되고 실수를 나누면 소수점까지 계산

몫과 나머지를 구하는 연산

소스 코드 예제4-2.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     double apple;           // 실수
06     int banana;             // 정수
07     int orange;             // 정수
08
```

❖ 나누기 연산자와 나머지 연산자 (2/2)

```
09     apple = 5.0 / 2.0;           // 실수와 실수의 나누기 연산
10     banana = 5 / 2;             // 정수와 정수의 나누기 연산
11     orange = 5 % 2;             // 정수와 정수의 나머지 연산(%)
12
13     printf("apple : %.1lf\n", apple);
14     printf("banana : %d\n", banana);
15     printf("orange : %d\n", orange);
16
17     return 0;
18 }
```



실행결과

```
apple : 2.5
banana : 2
orange : 1
```

❖ 증감 연산자 (1/2)

- 피연산자의 값을 1 증가시키거나 감소시킨다.

증감 연산자의 연산

소스 코드 예제 4-3.c

```
01 #include <stdio.h>
```

```
02
```

```
03 int main(void)
```

```
04 {
```

```
05     int a = 10, b = 10;
```

```
06
```

```
07     ++a;                // 변수의 값을 1만큼 증가
```

```
08     --b;                // 변수의 값을 1만큼 감소
```

```
09
```

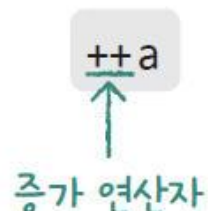
```
10     printf("a : %d\n", a);
```

```
11     printf("b : %d\n", b);
```

```
12
```

```
13     return 0;
```

```
14 }
```







❖ 증감 연산자 (2/2)

■ 전위 표기와 후위 표기

전위 표기와 후위 표기를 사용한 증감 연산

소스 코드 예제4-4.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 5, b = 5;
06     int pre, post;
07
08     pre = (++a) * 3;      // 전위형 증감 연산자
09     post = (b++) * 3;    // 후위형 증감 연산자
10
11     printf("초깃값 a = %d, b = %d\n", a, b);
12     printf("전위형: (++a) * 3 = %d, 후위형: (b++) * 3 = %d\n", pre, post);
13
14     return 0;
15 }
```

실행결과

초깃값 a = 6, b = 6

전위형: (++a) * 3 = 18, 후위형: (b++) * 3 = 15

❖ 관계 연산자 (1/2)

- 크다(>), 크거나 같다(>=), 작다(<)

관계 연산의 결과값 확인

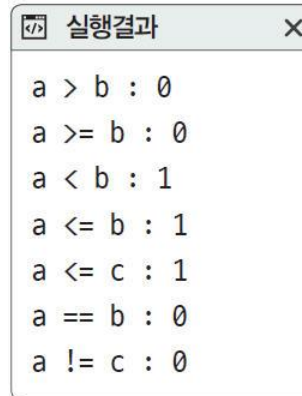
소스 코드 예제4-5.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10, b = 20, c = 10;
06     int res;                                // 결과값을 저장할 변수
07                                           // 각각 a와 b, c 값을 대입해보자.
08     res = (a > b);                          // 10 > 20은 거짓이므로 결과값은 0
09     printf("a > b : %d\n", res);
10     res = (a >= b);                        // 10 >= 20은 거짓이므로 결과값은 0
11     printf("a >= b : %d\n", res);
12     res = (a < b);                          // 10 < 20은 참이므로 결과값은 1
13     printf("a < b : %d\n", res);
```


❖ 관계 연산자 (2/2)

- 작거나 같다(<=), 같다(==), 같지 않다(!=)

```
14     res = (a <= b);           // 10 <= 20은 참이므로 결과값은 1
15     printf("a <= b : %d\n", res);
16     res = (a <= c);           // 10 <= 10은 참이므로 결과값은 1
17     printf("a <= c : %d\n", res);
18     res = (a == b);           // 10 == 20은 거짓이므로 결과값은 0
19     printf("a == b : %d\n", res);
20     res = (a != c);           // 10 != 10은 거짓이므로 결과값은 0
21     printf("a != c : %d\n", res);
22
23     return 0;
24 }
```



실행결과

```
a > b : 0
a >= b : 0
a < b : 1
a <= b : 1
a <= c : 1
a == b : 0
a != c : 0
```

❖ 논리 연산자 : &&(and), ||(or), !(not)

논리 연산의 결과값 확인

소스 코드 예제4-6.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 30;
06     int res;
07
08     res = (a > 10) && (a < 20);           // 좌항과 우항이 모두 참이면 참
09     printf("(a > 10) && (a < 20) : %d\n", res);
10     res = (a < 10) || (a > 20);           // 좌항과 우항 중 하나라도 참이면 참
11     printf("(a < 10) || (a > 20) : %d\n", res);
12     res = !(a >= 30);                     // 거짓이면 참으로, 참이면 거짓으로
13     printf("! (a >= 30) : %d\n", res);
14
15     return 0;
16 }
```

실행결과
(a > 10) && (a < 20) : 0
(a < 10) (a > 20) : 1
! (a >= 30) : 0

❖ 연산의 결과값을 처리하는 방법

- 연산의 결과는 저장하거나 바로 사용하지 않으면 사라진다.

연산의 결과값을 사용하는 방법

소스 코드 예제4-7.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10, b = 20, res;
06
07     a + b;                // 연산 결과는 버려짐
08     printf("%d + %d = %d\n", a, b, a + b); // 연산 결과를 바로 출력에 사용
09
10     res = a + b;          // 연산 결과를 변수에 저장
11     printf("%d + %d = %d\n", a, b, res);   // 저장된 값을 계속 사용
12
13     return 0;
14 }
```

실행결과
10 + 20 = 30
10 + 20 = 30

키워드로 끝내는 핵심 포인트

- ❖ **대입 연산자(=)**는 오른쪽 수식의 값을 왼쪽 변수에 저장한다.
- ❖ 두 값이 같은지를 확인할 때는 **관계 연산자**인 **==**를 사용한다.
- ❖ **나누기 연산자(/)**로 정수를 나누면 몫이 계산된다.
- ❖ 나머지는 **나머지 연산자(%)**로 연산한다.
- ❖ **증감 연산자**의 후위 표기는 변수의 값을 먼저 사용하고 증가
- ❖ **논리 연산**의 결과는 1(참) 또는 0(거짓)이 된다.
- ❖ **관계 연산**의 결과도 1(참) 또는 0(거짓)이 된다.

표로 정리하는 핵심 포인트

표 4-1 관계 연산자

연산식	결괏값
$a > b$	a가 b보다 크면 1(참, true), 그렇지 않으면 0(거짓, false)
$a \geq b$	a가 b보다 크거나 같으면 1(참), 그렇지 않으면 0(거짓)
$a < b$	a가 b보다 작으면 1(참), 그렇지 않으면 0(거짓)
$a \leq b$	a가 b보다 작거나 같으면 1(참), 그렇지 않으면 0(거짓)
$a == b$	a와 b가 같으면 1(참), 다르면 0(거짓)
$a != b$	a와 b가 다르면 1(참), 같으면 0(거짓)

표 4-2 논리 연산자

연산식	논리관계	결괏값
$a \&\& b$	논리곱(AND)	a와 b가 모두 참이면 1, 그렇지 않으면 0
$a \ \ b$	논리합(OR)	a와 b 중 하나라도 참이면 1, 그렇지 않으면 0
$!a$	논리부정(NOT)	a가 거짓이면 1, 참이면 0

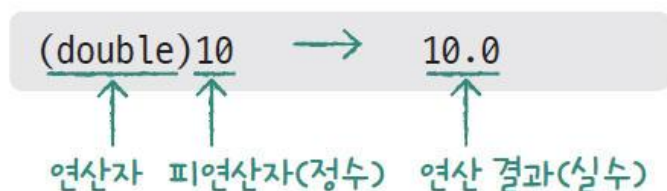
❖ 형 변환 연산자 (1/2)

- 연산할 때 피연산자의 형태가 다르면 자동으로 일치된다.

예) 정수 + 실수 => 실수 + 실수

- 형 변환 연산자는 피 연산자의 값을 원하는 형태로 변환시킨다.

정수를 실수로 바꾸는 경우



실수를 정수로 바꾸는 경우



❖ 형 변환 연산자 (2/2)

형 변환 연산자가 필요한 경우

소스 코드 예제4-8.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 20, b = 3;
06     double res;
07
08     res = ((double)a) / ((double)b);    // (double)을 사용해 a와 b의 값을 실수로 변환
09     printf("a = %d, b = %d\n", a, b);
10     printf("a / b의 결과 : %.1lf\n", res);
11
12     a = (int)res;                        // (int)를 사용해 res의 값에서 정수 부분만 추출
13     printf("(int) %.1lf의 결과 : %d\n", res, a);
14
15     return 0;
16 }
```

실행결과

a = 20, b = 3
a / b의 결과 : 6.7
(int) 6.7의 결과 : 6

❖ sizeof 연산자

sizeof 연산자의 사용 예

[소스 코드](#) 예제4-9.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10;
06     double b = 3.4;
07
08     printf("int형 변수의 크기 : %d\n", sizeof(a));
09     printf("double형 변수의 크기 : %d\n", sizeof(b));
10     printf("정수형 상수의 크기 : %d\n", sizeof(10));
11     printf("수식의 결과값의 크기 : %d\n", sizeof(1.5 + 3.4));
12     printf("char 자료형의 크기 : %d\n", sizeof(char));
13
14     return 0;
15 }
```

 실행결과 ×

```
int형 변수의 크기 : 4
double형 변수의 크기 : 8
정수형 상수의 크기 : 4
수식의 결과값의 크기 : 8
char 자료형의 크기 : 1
```

❖ 복합대입 연산자

복합대입 연산자 소스 코드 예제4-10.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10, b = 20;
06     int res = 2;
07
08     a += 20;           // a와 20을 더한 결과를 다시 a에 저장(+=)
09     res *= b + 10;     // b에 10을 더한 결과값에 res를 곱하고 다시 res에 저장(*=)
10
11     printf("a = %d, b = %d\n", a, b);
12     printf("res = %d\n", res);
13
14     return 0;
15 }
```

$$res* = b + 10$$

①
 $res * 30$ ← ①번 연산의 결과 = 30

②
 $res = 60$ ← ②번 연산의 결과 = 60

③ 대입 연산 후의 res의 값은 60

실행결과

a = 30, b = 20
res = 60

❖ 콤마 연산자

콤마 연산자 소스 코드 예제4-11.c

```

01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10, b = 20;
06     int res;
07
08     res = (++a, ++b);
09
10     printf("a:%d, b:%d\n", a, b);
11     printf("res:%d\n", res);
12
13     return 0;
14 }


```

② 남은 ++b을 연산해서 b 값이 1 증가

res = ++a, ++b;

① =와 ,의 우선순위 비교
=가 높으므로 res에 ++a 값을 저장

// 차례로 연산이 수행되며 결과적으로
// res에 저장되는 값은 증가된 b의 값이다.

 실행결과 ×

```

a:11, b:21
res:21

```

❖ 조건 연산자

조건 연산자

소스 코드 예제4-12.c

```
01 #include <stdio.h>
```

```
02
```

```
03 int main(void)
```

```
04 {
```

```
05     int a = 10, b = 20, res;
```

```
06
```

```
07     res = (a > b) ? a : b;    // a와 b 중에 큰 값이 res에 저장
```

```
08     printf("큰 값 : %d\n", res);
```

```
09
```

```
10     return 0;
```

```
11 }
```

① 10 > 20 결과값 false

② b 값 선정

res = (a > b) ? a : b; // 7행

③ b 값 대입



실행결과



큰 값 : 20

❖ 비트 연산자 (1/4)

비트 연산식의 결과 [소스 코드](#) 예제4-13.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10;           // 비트열 00000000 00000000 00000000 00001010
06     int b = 12;           // 비트열 00000000 00000000 00000000 00001100
07
08     printf("a & b : %d\n", a & b);
09     printf("a ^ b : %d\n", a ^ b);
10     printf("a | b : %d\n", a | b);
11     printf("~a : %d\n", ~a);
12     printf("a << 1 : %d\n", a << 1);
13     printf("a >> 2 : %d\n", a >> 2);
14
15     return 0;
16 }
```

실행결과

```
a & b : 8
a ^ b : 6
a | b : 14
~a : -11
a << 1 : 20
a >> 2 : 2
```

❖ 비트 연산자 (2/4)

- 비트별 논리곱 연산자(&)

	00000000	00000000	00000000	00001010	(a = 10)
&	00000000	00000000	00000000	00001100	(b = 12)
<hr/>					
	00000000	00000000	00000000	00001000	(a & b = 8)

- 비트별 배타적 논리합 연산자(^)

	00000000	00000000	00000000	00001010	(a = 10)
^	00000000	00000000	00000000	00001100	(b = 12)
<hr/>					
	00000000	00000000	00000000	00000110	(a ^ b = 6)

❖ 비트 연산자 (3/4)

- 비트별 논리합 연산자(|)

```

      00000000 00000000 00000000 00001010   (a = 10)
|) 00000000 00000000 00000000 00001100   (b = 12)
-----
      00000000 00000000 00000000 00001110   (a | b = 14)

```

- 비트별 부정 연산자(~)

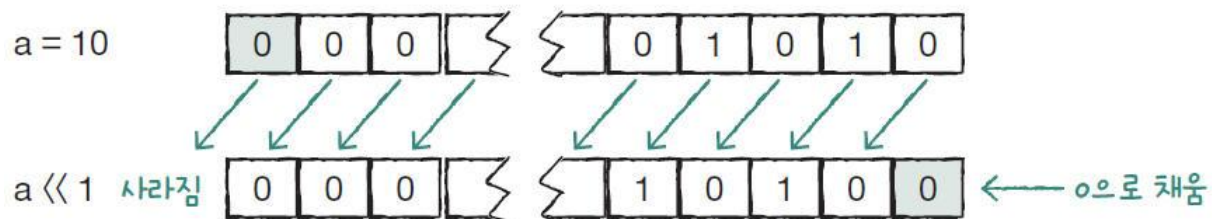
```

~) 00000000 00000000 00000000 00001010   (a = 10)
-----
      11111111 11111111 11111111 11110101   (~a = -11)

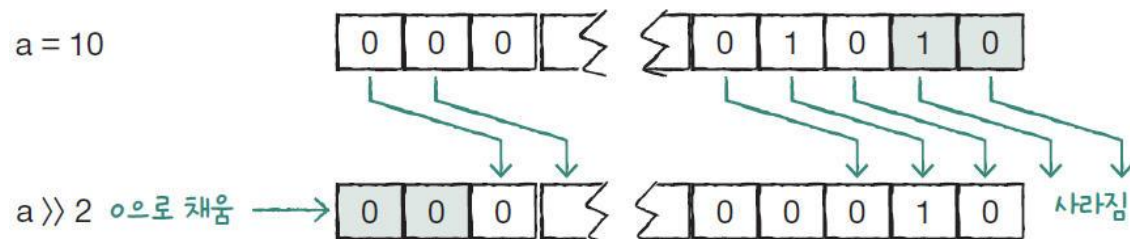
```


❖ 비트 연산자 (4/4)

- 비트별 왼쪽 이동 연산자(<<)



- 비트별 오른쪽 이동 연산자(>>)



❖ 연산자 우선순위와 연산 방향 (1/2)

연산자 우선순위와 연산 방향

소스 코드 예제 4-14.c

```
01 #include <stdio.h>
02
03 int main(void)
04 {
05     int a = 10, b = 5;
06     int res;
07
08     res = a / b * 2;           // 우선순위가 같으므로 왼쪽부터 차례로 연산
09     printf("res = %d\n", res);
10     res = ++a * 3;            // a의 값을 1증가시키고 3을 곱한다.
11     printf("res = %d\n", res);
12     res = a > b && a != 5;    // a > b의 결과와 a != 5의 결과를 && 연산
13     printf("res = %d\n", res);
14     res = a % 3 == 0;         // a % 3의 값이 0과 같은지 확인
15     printf("res = %d\n", res);
16
17     return 0;
18 }
```

실행결과

```
res = 4
res = 33
res = 1
res = 0
```

❖ 연산자 우선순위와 연산 방향 (2/2)

- 단항 연산자가 이항 연산자보다 우선순위가 높다.
- 산술 > 관계 > 논리 연산자 순서로 우선순위가 높다.

`++a * 3` // 10행

↑ ↑

① 단항 ② 이항

`a > b && a != 5` // 12행

↑ ↑ ↑

① 관계 ③ 논리 ② 관계

- 우선 순위가 같은 경우 연산 방향

우선순위가 같은 경우

`(a / b) * c`

대입 연산자

`a = (b = c)`

단항 연산자

`-((double) a)`

키워드로 끝내는 핵심 포인트

- ❖ **형 변환 연산자**는 피연산자의 값을 잠깐 원하는 형태로 바꾸나 변수의 형태는 바뀌지 않는다.
- ❖ **sizeof 연산자**는 괄호와 함께 사용하지만 함수는 아니다.
- ❖ **복합대입 연산자**의 우선순위는 대입 연산자와 같다.
- ❖ **비트 연산자**는 비트 단위로 연산하며 비트 논리 연산자(&, ^, |)와 비트 이동 연산자(>>, <<)가 있다.

표로 정리하는 핵심 포인트 (1/3)

표 4-3 연산자의 종류와 우선순위

종류	우선순위	연산자(괄호의 숫자는 우선순위)	연산 방향
1차 연산자	1	() [] . ->	→
단항 연산자	2	- ++ -- ~ ! * & sizeof (type)	←
산술 연산자	3	* / %	→
	4	+ -	
비트 이동 연산자	5	<< >>	
관계 연산자	6	< <= > >=	
동등 연산자	7	== !=	
비트 논리 연산자	8	&	
	9	^	
	10		
논리 연산자	11	&&	→
	12		
조건 연산자	13	?:	←
대입 연산자	14	= += -= *= /= %= &= ^= = <<= >>=	
coma 연산자	15	,	→

표로 정리하는 핵심 포인트 (2/3)

표 4-4 기타 연산자

연산자	연산식 예	결괏값
형 변환 연산자	<code>res = (int)10.7;</code>	res 값은 10
sizeof 연산자	<code>res = sizeof(double);</code>	res 값은 8
복합대입 연산자	<code>a += 10;</code>	a의 값을 10 증가
coma 연산자	<code>res = (a , b);</code>	res에 b 값 저장
조건 연산자	<code>res = (a > b) ? a : b;</code>	a가 b보다 크면 res 값은 a 작거나 같으면 res 값은 b
비트 연산자	<code>a & b; ~a; a << b;</code>	a와 b의 비트 상태에 따라 결괏값이 다름

표 4-5 복합대입 연산자

복합대입 연산식	동일한 연산식	복합대입 연산식	동일한 연산식
<code>a += b</code>	<code>a = a + b</code>	<code>a &= 2</code>	<code>a = a & 2</code>
<code>a -= b</code>	<code>a = a - b</code>	<code>a ^= 2</code>	<code>a = a ^ 2</code>
<code>a *= b</code>	<code>a = a * b</code>	<code>a = 2</code>	<code>a = a 2</code>
<code>a /= b</code>	<code>a = a / b</code>	<code>a <<= 2</code>	<code>a = a << 2</code>
<code>a %= b</code>	<code>a = a % b</code>	<code>a >>= 2</code>	<code>a = a >> 2</code>

표로 정리하는 핵심 포인트 (3/3)

표 4-6 비트 연산자 종류

구분	연산자	연산 기능
비트 논리 연산자	&	비트 단위 논리곱(AND) 연산자
		& 연산은 두 비트가 모두 1인 경우에만 1로 계산한다.
	^	비트 단위 배타적 논리합(XOR) 연산자
		^ 연산은 두 비트가 서로 다른 경우만 1로 계산한다.
		비트 단위 논리합(OR) 연산자
		연산은 두 비트 중에서 하나라도 참이면 1로 계산한다.
	~	비트 단위 부정(NOT) 연산자
		~ 연산은 1은 0으로 바꾸고 0은 1로 바꾼다.
비트 이동 연산자	<<	왼쪽 비트 이동 연산자
		<< 연산자는 왼쪽으로 이동시킨다.
	>>	오른쪽 비트 이동 연산자
		>> 연산자는 오른쪽으로 이동시킨다.