

Chapter 04

C 연산자의 이해



목차

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

01

산술 연산자

1. 산술 연산자

1. 기본 산술 연산자

연산자	설명	사용 예	사용 예에 대한 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 5와 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.
%	나머지 값	$a = 5 \% 3$	정수 5를 3으로 나눈 뒤 나머지 값을 a에 대입한다.

1. 산술 연산자

1. 기본 산술 연산자

기본 4-1 산술 연산자 사용 예

4-1.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a, b = 5, c = 3;
06
07     a = b + c;      ——— 더하기 연산을 해서 a에 대입한다.
08     printf(" %d + %d = %d \n", b, c, a);
09
10     a = b - c;      ——— 빼기 연산을 해서 a에 대입한다.
11     printf(" %d - %d = %d \n", b, c, a);
12
13     a = b * c;      ——— 곱하기 연산을 해서 a에 대입한다.
14     printf(" %d * %d = %d \n", b, c, a);
15
16     a = b / c;      ——— 나누기 연산을 해서 a에 대입한다.
17     printf(" %d / %d = %d \n", b, c, a);
18
19     a = b % c;      ——— 나머지값 연산을 해서 a에 대입한다.
20     printf(" %d %% %d = %d \n", b, c, a);
21 }
```

실행 결과

5 + 3 = 8
5 - 3 = 2
5 * 3 = 15
5 / 3 = 1
5 % 3 = 2

1. 산술 연산자

2. 연산자 우선순위와 강제 형 변환

응용 4-2 연산자 우선순위와 강제 형 변환 예

4-2.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 2, b = 3, c = 4;      — 정수형 변수를 선언한다.
06     int result1, mok, namugi;
07     float result2;              — 실수형 변수를 선언한다.
08
09     result1 = a + b - c;         — 더하기와 빼기 연산을 동시에 수행한다.
10     printf(" %d + %d - %d = %d \n", a, b, c, result1);
11
12     result1 = a + b * c;         — 더하기와 곱하기 연산을 동시에 수행한다.
13     printf(" %d + %d * %d = %d \n", a, b, c, result1);
14
15     result2 = a * b / (float) c; — 정수 c를 실수로 강제 형 변환한 후 연산한다.
16     printf(" %d * %d / %d = %f \n", a, b, c, result2);
17
18     ❶ = c / b;                  — 몫을 구한다.
19     printf(" %d / %d 의 몫은 %d \n", c, b, mok);
20
21     ❷ = c % b;                  — 나머지를 구한다.
22     printf(" %d %d 의 나머지는 %d \n", c, b, namugi);
23 }
```

실행 결과

$$2 + 3 - 4 = 1$$

$$2 + 3 * 4 = 14$$

$$2 * 3 / 4 = 1.500000$$

$$4 / 3 \text{ 의 몫은 } 1$$

$$4 / 3 \text{ 의 나머지는 } 1$$

1. 산술 연산자

2. 연산자 우선순위와 강제 형 변환

▪ 간단한 연산자 우선순위

- [응용 4-2]의 5행과 6행에서 정수형 변수를 선언하고 7행에서 실수형 결과를 저장할 실수형 변수 result2를 선언

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

- 답은 ❶이지만 덧셈과 뺄셈의 경우에는 계산되는 순서(연산자 우선순위)가 동일하므로 어떤 것을 먼저 계산하든 결과가 같음
- 괄호가 없을 때는 왼쪽에서 오른쪽 방향으로 계산
- 덧셈과 곱셈이 같이 있는 12행을 보면 무엇을 먼저 계산하느냐에 따라 결과가 다르게 나옴

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

1. 산술 연산자

2. 연산자 우선순위와 강제 형 변환

- 데이터 형식의 강제 형 변환

여기서 잠깐 괄호를 사용한 연산자 우선순위

- 덧셈, 뺄셈, 곱셈, 나눗셈이 함께 나와 연산자 우선순위가 혼란스러울 때는 괄호를 사용
- 다음 ❶과 ❷은 동일한 결과를 출력하지만 ❷가 더 나은 코딩이라고 할 수 있음
- ❶ $a = b + c * d ;$
- ❷ $a = b + (c * d);$

1. 산술 연산자

2. 연산자 우선순위와 강제 형 변환

- 데이터 형식의 강제 형 변환

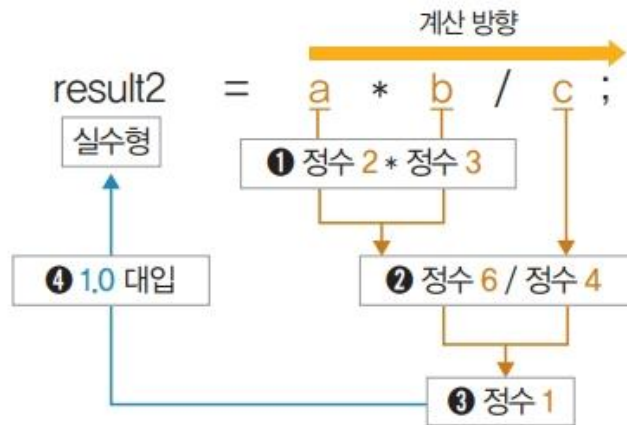


그림 4-1 강제 형 변환을 하지 않았을 때의 결과

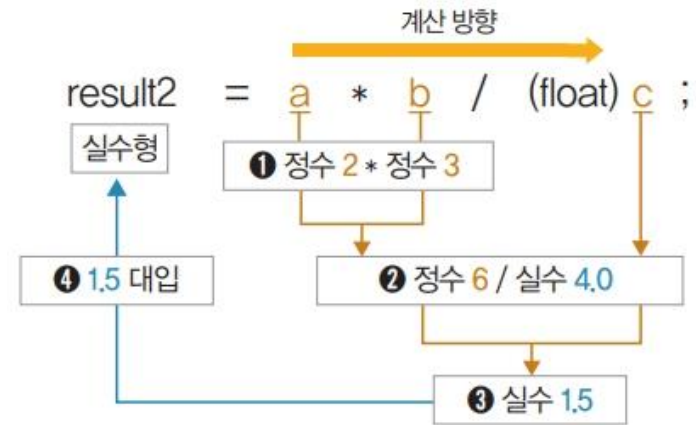


그림 4-2 강제 형 변환을 했을 때의 결과

1. 산술 연산자

3. 대입 연산자와 증감 연산자

- C에서는 대입 연산자 = 외에도 +=, -=, *=, /=, %=를 사용할 수 있음
- 값을 1씩 증가 시키는 역할을 하는 ++ 연산자와 1씩 감소시키는 역할을 하는 -- 연산자도 있음

표 4-2 대입 연산자와 증감 연산자

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.
++	증가 연산자	a++ 또는 ++a	a += 1 또는 a = a + 1과 동일하다.
--	감소 연산자	a-- 또는 --a	a -= 1 또는 a = a - 1과 동일하다.

1. 산술 연산자

3. 대입 연산자와 증감 연산자

기본 4-3 대입 연산자와 증감 연산자 사용 예

4-3.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 10;
06
07     a++;           ----- a = a + 1과 동일하다.
08     printf(" a ++ ==> %d \n", a);
09
10     a--;           ----- a = a - 1과 동일하다.
11     printf(" a -- ==> %d \n", a);
12
13     a += 5;        ----- a = a + 5와 동일하다.
14     printf(" a += 5 ==> %d \n", a);
15
16     a -= 5;        ----- a = a - 5와 동일하다.
17     printf(" a -= 5 ==> %d \n", a);
18
```

1. 산술 연산자

3. 대입 연산자와 증감 연산자

```
19  a *= 5; ----- a = a * 5와 동일하다.
20  printf("a *= 5 ==> %d \n", a);
21
22  a /= 5; ----- a = a / 5와 동일하다.
23  printf("a /= 5 ==> %d \n", a);
24
25  a %= 5; ----- a = a % 5와 동일하다.
26  printf("a %= 5 ==> %d \n", a);
27 }
```

실행 결과

```
a ++ ==> 11
a -- ==> 10
a += 5 ==> 15
a -= 5 ==> 10
a *= 5 ==> 50
a /= 5 ==> 10
a %= 5 ==> 0
```

1. 산술 연산자

3. 대입 연산자와 증감 연산자

- `a++`는 'a가 있고 a 값을 1 증가시켜라'라는 의미 이고, `++a`는 'a 값을 1 증가시키고 a가 있다'라는 의미

응용 4-4 증감 연산자 사용 예

4-4.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 10, b;
06
07     b = a++;           — b=a를 수행한 후 a를 1 증가시킨다.
08     printf(" %d \n", b);
09
10     a++;               — a를 1 증가시킨 후 b=a를 수행한다.
11     printf(" %d \n", b);
12 }
```

:e++ = q

실행 결과

```
10
12
```

02

관계 연산자

2. 관계 연산자

- 관계 연산자(또는 비교 연산자)는 어떤 것이 큰지, 작은지, 같은지를 비교하는 것으로 그 결과는 참(true)이나 거짓(false) 중 하나
- 조건문(if)이나 반복문(for, while)에서 사용하며 단독으로 사용하는 경우는 별로 없음
- 일반적으로 참은 1로, 거짓은 0으로 표시

$$a < b = \begin{cases} \text{참: 1} \\ \text{거짓: 0} \end{cases}$$

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. 관계 연산자

기본 4-5 관계 연산자 사용 예

4-5.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 100 , b = 200;
06
07     printf(" %d == %d 는 %d 이다.\n", a, b, a == b);
08     printf(" %d != %d 는 %d 이다.\n", a, b, a != b);
09     printf(" %d > %d 는 %d 이다.\n", a, b, a > b);
10     printf(" %d < %d 는 %d 이다.\n", a, b, a < b);
11     printf(" %d >= %d 는 %d 이다.\n", a, b, a >= b);
12     printf(" %d <= %d 는 %d 이다.\n", a, b, a <= b);
13
14     printf(" %d = %d 는 %d 이다.\n", a, b, a=b);
15 }
```

—— 관계 연산자 '같다'

—— 관계 연산자 '같지 않다'

—— 관계 연산자 '크다'

—— 관계 연산자 '작다'

—— 관계 연산자 '크거나 같다'

—— 관계 연산자 '작거나 같다'

—— 관계 연산자가 아닌
대입 연산자를 수행한다.

실행 결과

100 == 200 는 0 이다.
100 != 200 는 1 이다.
100 > 200 는 0 이다.
100 < 200 는 1 이다.
100 >= 200 는 0 이다.
100 <= 200 는 1 이다.
200 = 200 는 200 이다.

2. 관계 연산자

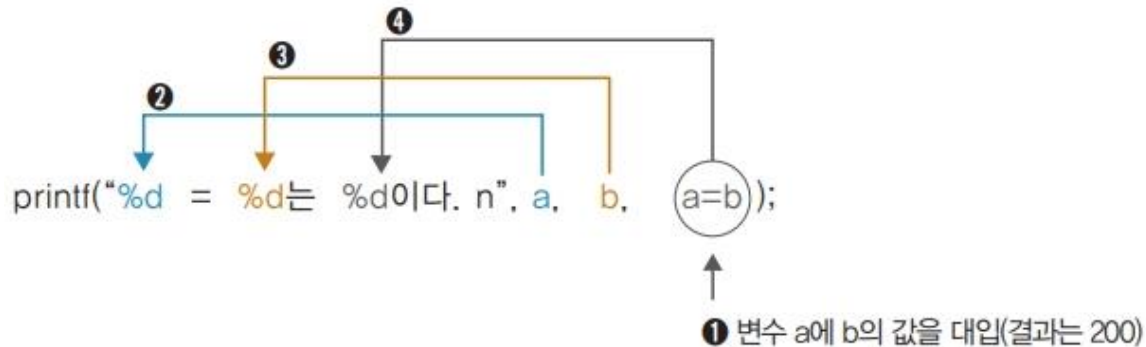


그림 4-5 대입 연산자의 작동

- ①의 $a=b$ 는 b의 값을 a에 대입하라는 의미
- 현재 b에는 200이 들어 있으므로 a에도 200이 대입
- 그 결과 $a=b$ 의 위치에는 200이라는 값이 들어감
- ②의 a에는 현재 200이 들어 있으므로 첫 번째 %d에는 200이 대입
- ③의 b에도 현재 200이 들어 있으므로 두 번째 %d 에도 200이 대입
- ④의 $a=b$ 는 200이므로 세 번째 %d에도 200이 대입되어 결국 '200 = 200는 200이다.'가 출력

03

논리 연산자

3. 논리 연산자

- 논리 연산자는 주로 여러 가지 조건을 복합적으로 사용하며 &&(그리고), ||(또는), !(부정)가 쓰임
- 예를 들어 a라는 값이 100과 200 사이에 들어 있어야 한다면 'a는 100보다 크다. 그리고 a는 200보다 작다.'라고 표현할 수 있음

```
(a > 100 ) && (a < 200)
```

- 참이 되려면 (a > 100)도 참이 되어야 하고 (a < 200)도 참이 되어야 함
- 즉 a가 100과 200 사이에 있어야만 두 조건 모두 참이 됨

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. 논리 연산자

기본 4-6 논리 연산자 사용 예 1

4-6.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 99;
06
07     printf(" AND 연산 : %d \n", (a >= 100) && (a <= 200)); ----- AND 연산을 사용한다.
08     printf(" OR 연산 : %d \n", (a >= 100) || (a <= 200)); ----- OR 연산을 사용한다.
09     printf(" NOT 연산 : %d \n", !(a==100)); ----- NOT 연산을 사용한다.
10 }
```

실행 결과

AND 연산 : 0
OR 연산 : 1
NOT 연산 : 1

3. 논리 연산자

응용 4-7 논리 연산자 사용 예 2

4-7.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 100, b = -200;
06
07     printf(" 상수의 AND 연산 : %d \n", a && b);
08     printf(" 상수의 OR 연산 : %d \n", __1__);
09     printf(" 상수의 NOT 연산 : %d \n", __2__);
10 }
```

----- AND 연산을 사용한다.

----- OR 연산을 사용한다.

----- NOT 연산을 사용한다.

이 코드를 실행하면

실행 결과

상수의 AND 연산 : 1
상수의 OR 연산 : 1
상수의 NOT 연산 : 0

04

비트 연산자

4. 비트 연산자

- 비트 연산자는 정수나 문자 등을 2진수로 변환한 후 각 자리의 비트끼리 연산을 수행

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.

4. 비트 연산자

■ 비트 논리곱(&) 연산자

- [그림 4-6]에서 보이듯이 10진수를 2진수로 변환한 후 각 비트마다 AND 연산을 수행

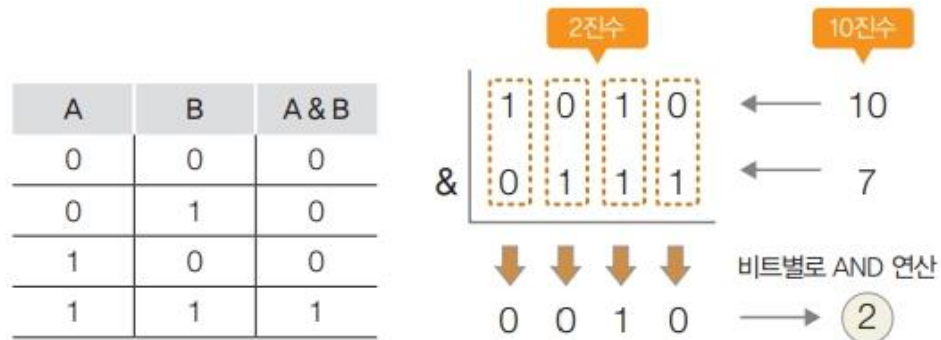


그림 4-6 비트 논리곱의 예

기본 4-8 비트 논리곱 연산자 사용 예

4-8.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     printf(" 10 & 7 = %d \n", 10 & 7);
06     printf(" 123 & 456 = %d \n", 123 & 456);
07     printf(" 0xFFFF & 0000 = %d \n ", 0xFFFF & 0000);
08 }
```

—— 10과 7의 비트 논리곱을 수행한다.

—— 123과 456의 비트 논리곱을 수행한다.

—— 16진수 FFFF와 0의 비트 논리곱을 수행한다.

4. 비트 연산자

■ 비트 논리곱(&) 연산자

실행 결과

$10 \& 7 = 2$

$123 \& 456 = 72$

$0xFFFF \& 0000 = 0$

- [기본 4-8] 5행의 10과 7의 연산은 [그림 4-6]과 동일
- 6행에서는 123의 2진수인 1111011_2 과 456의 2진수인 111001000_2 의 비트 논리곱 결과가 1001000_2 이므로 10진수로 72
- 7행에서는 16진수 FFFF(2진수로는 1111 1111 1111 1111)와 0000(2진수로는 0000 0000 0000 0000)의 비트 논리곱 결과인 0이 출력
- 0과 비트 논리곱을 수행하면 무조건 0이 나온다는 것을 기억할 것

4. 비트 연산자

■ 비트 논리합(|) 연산자

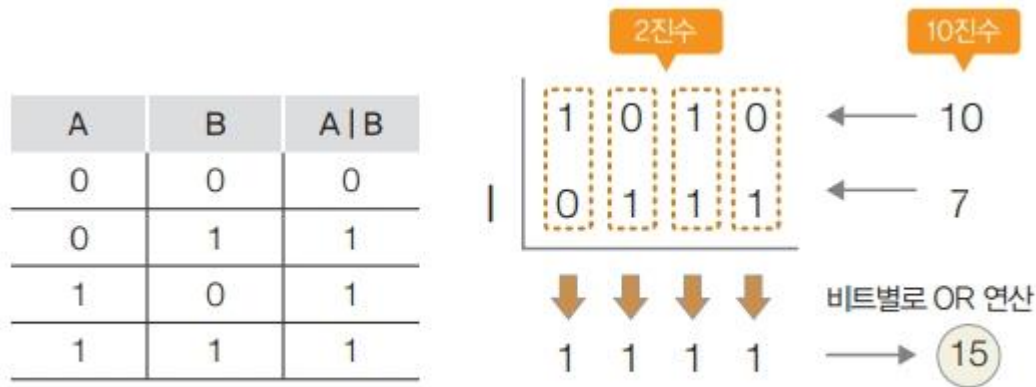


그림 4-7 비트 논리합의 예

기본 4-9 비트 논리합 연산자 사용 예

4-9.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     printf(" 10 | 7 = %d \n", 10 | 7);
06     printf(" 123 | 456 = %d \n", 123 | 456);
07     printf(" 0xFFFF | 0000 = %d \n ", 0xFFFF | 0000);
08 }
```

—— 10과 7의 비트 논리합을 수행한다.

—— 123과 456의 비트 논리합을 수행한다.

—— 16진수 FFFF와 0의 비트 논리합을 수행한다.

4. 비트 연산자

■ 비트 논리합(|) 연산자

실행 결과

10 | 7 = 15

123 | 456 = 507

0xFFFF | 0000 = 65535

■ 비트 논리합(^) 연산자

- 비트 배타적 논리합(^)은 두 값이 다르면 1, 같으면 0이 됨
- '참(1) ^ 참(1)'이나 '거짓(0) ^ 거짓(0)'이면 결과가 거짓(0)이고, '참(1) ^ 거짓(0)'이나 '거짓(0) ^ 참(1)'이면 결과가 참(1)

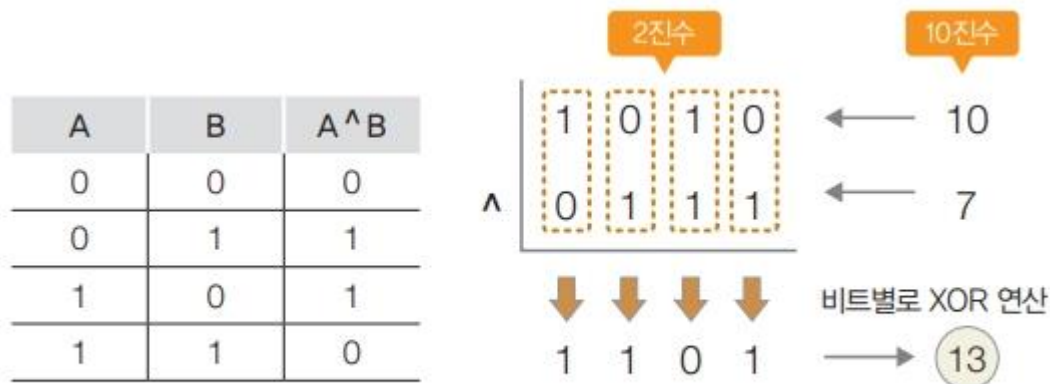


그림 4-8 비트 배타적 논리합의 예

4. 비트 연산자

■ 비트 논리합(^) 연산자

기본 4-10 비트 배타적 논리합 연산자 사용 예

4-10.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     printf(" 10 ^ 7 = %d \n", 10 ^ 7);      —— 10과 7의 비트 배타적 논리합을 수행한다.
06     printf(" 123 ^ 456 = %d \n", 123 ^ 456); —— 123과 456의 비트 배타적 논리합을 수행한다.
07     printf(" 0xFFFF ^ 0000 = %d \n ", 0xFFFF ^ 0000);
08 }                                             —— 16진수 FFFF와 0의 비트 배타적 논리합을 수행한다.
```

실행 결과

```
10 ^ 7 = 13
123 ^ 456 = 435
0xFFFF ^ 0000 = 65535
```

4. 비트 연산자

■ 비트 논리합(^) 연산자

응용 4-11 비트 연산에 마스크를 사용한 예

4-11.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     char a = 'A', b, c;
06     char mask = 0x0F;          ----- 마스크 값(0000 11112)을 설정한다.
07
08     printf(" %X & %X = %X \n", a, mask, a & mask); ----- 'A'와 0x0F의 비트 논리곱을 수행한다.
09     printf(" %X | %X = %X \n", a, mask, a | mask) ----- 'A'와 0x0F의 비트 논리합을 수행한다.
10
11     mask = 'a' - 'A';          ----- 'a'와 'A'의 차이는 32이다.
12
13     b = __1__                  ----- 'A'와 마스크(32)의 비트 배타적 논리합을 수행한다.
14     printf(" %c ^ %d = %c \n", a, mask, b);
15     a = __2__                  ----- 'a'와 마스크(32)의 비트 배타적 논리합을 수행한다.
16     printf(" %c ^ %d = %c \n", b, mask, a);
17 }
```

4. 비트 연산자

■ 비트 논리합(^) 연산자

자판: 1 a v mask; 2 b v mask;

실행 결과

41 & F = 1

41 & $F = 4F$

$$A^{32} = a$$
$$a^{32} = A$$

- 우선 6행에서 마스크 값을 16진수 $0x0F_{16}$ 로 선언했는데 이는 2진수로 00001111_2
- 비트 논리곱 연산을 진행하면 앞의 4비트는 모두 0000이 되고 뒤의 4비트는 A의 원래 값이 그대로 남음

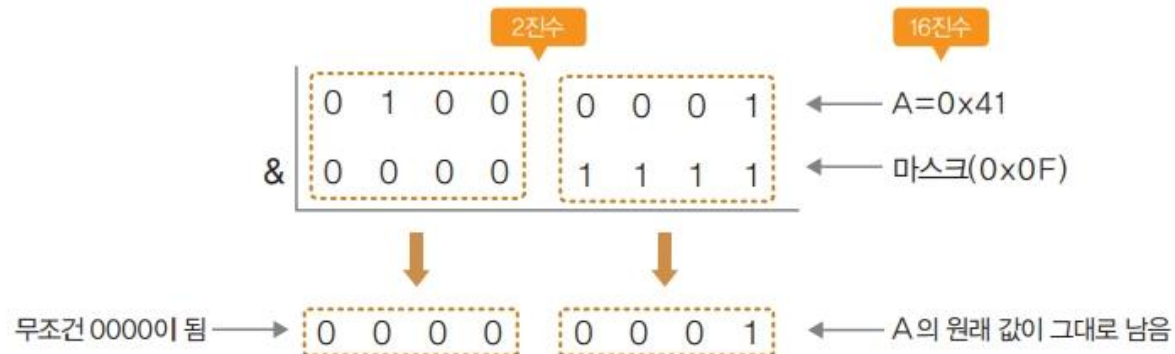


그림 4-9 마스크 0x0F를 사용한 비트 논리곱의 예

4. 비트 연산자

■ 비트 논리합(^) 연산자

- 9행의 0x0F로 비트 논리합 연산을 하면 [그림 4-10]과 같이 앞의 4비트는 A의 원래 값이 그대로 남고 뒤의 4비트는 무조건 1111

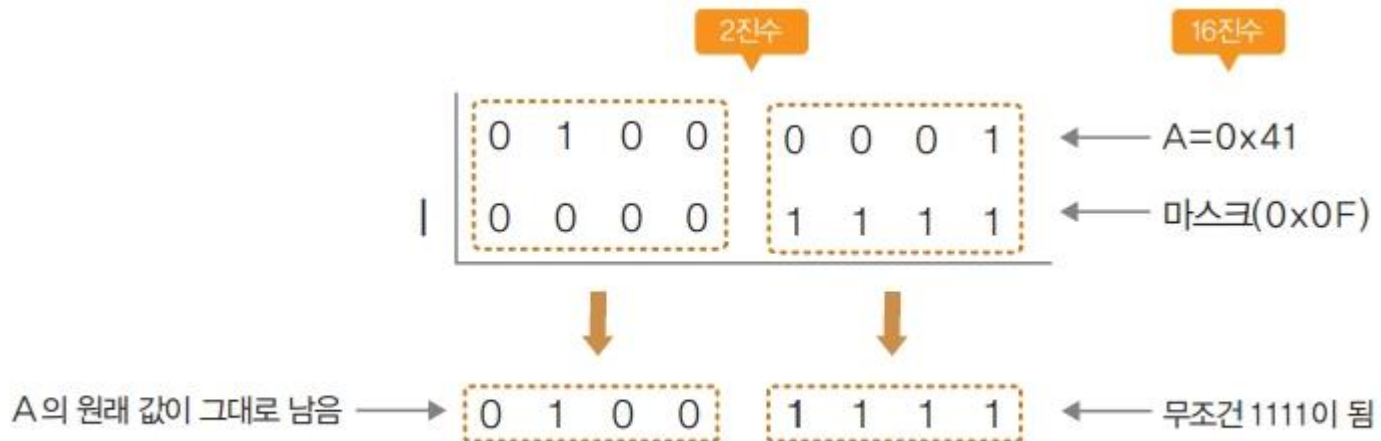


그림 4-10 마스크 0x0F를 사용한 비트 논리합의 예

4. 비트 연산자

■ 비트 부정(~) 연산자

- 비트 부정 연산자(또는 보수 연산자)는 두 수를 연산하는 것이 아니라 하나의 수를 가지고 각 비트를 반대로 만드는 연산자
- 모든 0은 1로, 모든 1은 0으로 바꿈
- 이렇게 반전된 값을 '1의 보수'라고 하며, 그 값에 1을 더한 값을 '2의 보수'라고 함

기본 4-12 비트 부정 연산자 사용 예

4-12.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 12345;
06
07     printf(" %d \n", ~a + 1);
08 }
```

—— 2의 보수(a 값)를 구한다.

실행 결과

-12345

4. 비트 연산자

■ 비트 왼쪽 시프트(<<) 연산자

- 비트 왼쪽 시프트 연산자는 나열된 비트를 왼쪽으로 시프트(shift)하는 연산자



그림 4-11 26을 왼쪽으로 두 칸 시프트한 결과

- 왼쪽으로 두 칸 이동했으므로 앞의 두 00이 떨어져 나가고 비어 있는 뒤의 두 칸에는 00이 채워짐
- 왼쪽으로 1회 시프트할 때는 2^1 을, 2회 시프트할 때는 2^2 을, 3회 시프트 할 때는 2^3 을 곱하는 셈

4. 비트 연산자

■ 비트 왼쪽 시프트(<<) 연산자

기본 4-13 비트 왼쪽 시프트 연산자 사용 예

4-13.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 10;
06     printf("%d 를 왼쪽 1회 시프트하면 %d 이다.\n", a, a << 1);
07     printf("%d 를 왼쪽 2회 시프트하면 %d 이다.\n", a, a << 2);
08     printf("%d 를 왼쪽 3회 시프트하면 %d 이다.\n", a, a << 3);
09 }
```

———— 왼쪽으로 시프트한 결과를
출력한다.

실행 결과

10 를 왼쪽 1회 시프트하면 20 이다.
10 를 왼쪽 2회 시프트하면 40 이다.
10 를 왼쪽 3회 시프트하면 80 이다.

4. 비트 연산자

■ 비트 오른쪽 시프트(>>) 연산자

- 비트 오른쪽 시프트 연산자는 나열된 비트를 오른쪽으로 시프트하는 연산자

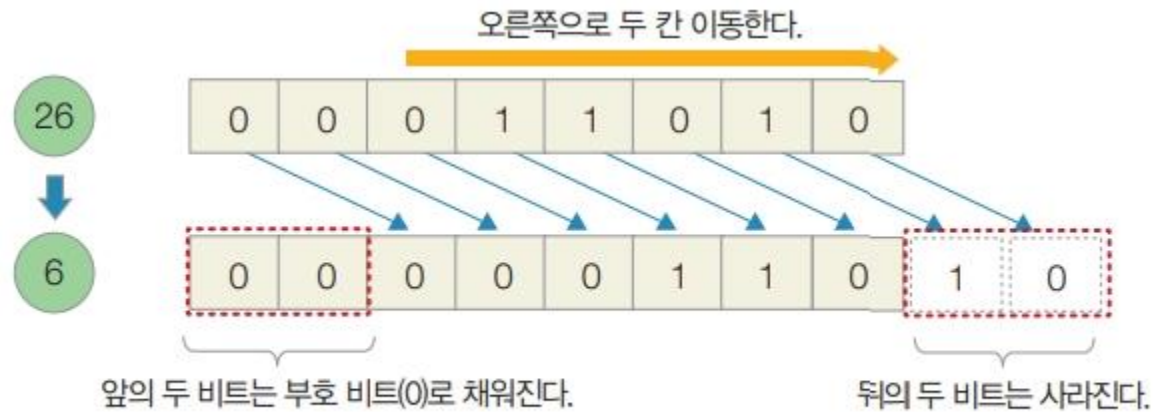


그림 4-12 26을 오른쪽으로 두 칸 시프트한 결과

- 오른쪽으로 1회 시프트할 때는 2^1 으로, 2회 시프트할 때는 2^2 으로, 3회 시프트할 때는 2^3 으로 나누는 원리

4. 비트 연산자

■ 비트 오른쪽 시프트(>>) 연산자

기본 4-14 비트 오른쪽 시프트 연산자 사용 예

4-14.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 10;
06     printf("%d 를 오른쪽 1회 시프트하면 %d 이다.\n", a, a >> 1);
07     printf("%d 를 오른쪽 2회 시프트하면 %d 이다.\n", a, a >> 2);
08     printf("%d 를 오른쪽 3회 시프트하면 %d 이다.\n", a, a >> 3);
09     printf("%d 를 오른쪽 4회 시프트하면 %d 이다.\n", a, a >> 4);
10 }
```

—— 오른쪽으로 시프트한
결과를 출력한다.

실행 결과

10 를 오른쪽 1회 시프트하면 5 이다.
10 를 오른쪽 2회 시프트하면 2 이다.
10 를 오른쪽 3회 시프트하면 1 이다.
10 를 오른쪽 4회 시프트하면 0 이다.

4. 비트 연산자

- **비트 오른쪽 시프트(>>) 연산자**

응용 4-15 비트 왼쪽 시프트, 비트 오른쪽 시프트 연산자 사용 예

4-15.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05     int a = 100, result;
06     int i;
07
08     for(i = 1; i <= 5; i ++ )
09     {
10         result = __1__
11         printf("%d << %d = %d\n", a, i, result);
12     }
13
14     for(i = 1; i <= 5; i ++ )
15     {
16         result = __2__
17         printf("%d >> %d = %d\n", a, i, result);
18     }
19 }
```

- 왼쪽 시프트 연산을 5회 반복해서 출력한다.

- 오른쪽 시프트 연산을 5회 반복해서 출력한다.

실행 결과

```
100 << 1 = 200
100 << 2 = 400
100 << 3 = 800
100 << 4 = 1600
100 << 5 = 3200
100 >> 1 = 50
100 >> 2 = 25
100 >> 3 = 12
100 >> 4 = 6
100 >> 5 = 3
```

05

연산자 우선순위

5. 연산자 우선순위

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	,	coma 연산자	➡

*

예제 모음

[예제모음 08] 입력된 두 실수의 산술 연산

예제 설명 실수를 입력받아 두 수의 다양한 연산을 출력하는 프로그램이다.

힌트 나머지를 구할 때는 강제 형 변환을 사용한다.

실행 결과

첫 번째 계산할 값을 입력하세요 ==> 10

두 번째 계산할 값을 입력하세요 ==> 20

$10.00 + 20.00 = 30.00$

$10.00 - 20.00 = -10.00$

$10.00 * 20.00 = 200.00$

$10.00 / 20.00 = 0.50$

$10 \% 20 = 10$

[예제모음 08] 입력된 두 실수의 산술 연산

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05     float a, b;           —— 실수형 변수를 선언한다.
06     float result;
07
08     printf("첫 번째 계산할 값을 입력하세요 ==> ");
09     scanf("%f", &a);       —— 실수를 입력받는다.
10     printf("두 번째 계산할 값을 입력하세요 ==> ");
11     scanf("%f", &b);       —— 실수를 입력받는다.
12
13     result = a + b;        —— 실수의 덧셈을 수행한다.
14     printf(" %5.2f + %5.2f = %5.2f \n", a, b, result);
15     result = a - b;        —— 실수의 뺄셈을 수행한다.
16     printf(" %5.2f - %5.2f = %5.2f \n", a, b, result);
17     result = a * b;        —— 실수의 곱셈을 수행한다.
18     printf(" %5.2f * %5.2f = %5.2f \n", a, b, result);
19     result = a / b;        —— 실수의 나눗셈을 수행한다.
20     printf(" %5.2f / %5.2f = %5.2f \n", a, b, result);
21     result = (int)a % (int)b; —— 나머지 연산을 위해 실수를 정수로 강제 형 변환한다.
22     printf(" %d %s %d = %d \n", (int)a, (int)b, (int)result);
23 }
```

[예제모음 09] 동전 교환 프로그램

예제 설명 입력된 액수만큼 500원, 100원, 50원, 10원짜리 동전으로 교환해주는 프로그램이다.

❶ 동전의 총 개수를 최소화한다.

❷ 고액의 동전을 먼저 바꿔준다.

실행 결과

```
## 교환할 돈은 ? 7777
```

```
오백 원짜리 => 15 개
```

```
백 원짜리   => 2 개
```

```
오십 원짜리 => 1 개
```

```
십 원짜리   => 2 개
```

```
바꾸지 못한 잔돈 => 7 원
```

[예제모음 09] 동전 교환 프로그램

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05     int money, c500, c100, c50, c10;  —— 입력한 돈과 각 동전의 개수를 저장하는 변수이다.
06
07     printf(" ## 교환할 돈은 ? ");
08     scanf("%d", &money);  —— 동전으로 교환할 액수를 입력한다.
09
10     c500 = money / 500;  —— 500원짜리 동전의 개수를 계산한다.
11     money = money % 500;  —— 500원짜리 동전으로 바꾼 후 나머지 금액이다.
12
13     c100 = money / 100;  —— 100원짜리 동전의 개수를 계산한다.
14     money = money % 100;  —— 100원짜리 동전으로 바꾼 후 나머지 금액이다.
15
16     c50 = money / 50;  —— 50원짜리 동전의 개수를 계산한다.
17     money = money % 50;  —— 50원짜리 동전으로 바꾼 후 나머지 금액이다.
18
19     c10 = money / 10;  —— 10원짜리 동전의 개수를 계산한다.
20     money = money % 10;  —— 10원짜리 동전으로 바꾼 후 나머지 금액이다.
21
22     printf("\n 오백 원짜리 ==> %d 개 \n", c500);
23     printf(" 백 원짜리 ==> %d 개 \n", c100);
24     printf(" 오십 원짜리 ==> %d 개 \n", c50);
25     printf(" 십 원짜리 ==> %d 개 \n", c10);
26     printf(" 바꾸지 못한 잔돈 ==> %d 원 \n", money);
27 }
```

—— 바꾸지 못한 나머지 돈은 money에 들어 있다.

[예제모음 10] 윤년 계산 프로그램

예제 설명 입력된 연도가 윤년인지 계산하는 프로그램이다.

- ❶ 4로 나누어 떨어지고 100으로 나누어 떨어지지 않으면 윤년이다.
- ❷ 400으로 나누어 떨어지는 해도 윤년에 포함된다.

실행 결과

연도를 입력하세요. : 2024
2024 년은 윤년입니다.

[예제모음 10] 윤년 계산 프로그램

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05     int year;
06
07     printf("연도를 입력하세요. : ");
08     scanf("%d", &year);      —— 계산할 연도를 입력한다.
09
10     if( ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0) )
11         printf("%d 년은 윤년입니다. \n", year);      — 윤년은 입력한 연도가 4로 나누어 떨어지고
12     else                                           100으로는 나누어 떨어지지 않아야 한다.
13         printf("%d 년은 윤년이 아닙니다. \n", year);   또는 400으로 나누어 떨어져도 된다.
14 }
```

감사합니다!

