

자료형과 연산 - 1



2023. Spring

국민대학교 소프트웨어학부
최 준수

비트단위 연산자

- 비트단위(bitwise) 연산자
 - 종류 및 연산

비트단위 연산자	설명
	비트단위 OR 연산자
&	비트단위 AND 연산자
^	비트단위 XOR 연산자
~	비트단위 NOT 연산자 (단항 연산자)
<<	비트단위 왼쪽 시프트 연산자
>>	비트단위 오른쪽 시프트 연산자

a	b	a b	a & b	a ^ b	~a
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

비트단위 OR 연산자

- 비트단위(bitwise) OR 연산자
 - 피연산자: 정수
 - 피연산자의 위치가 같은 비트들에 대해서 비트단위 OR 연산을 수행
 - 예

m	0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1	0x6be9 (unsigned short)
n	1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 1	0xfa9b (unsigned short)
m n	1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1	0xfbfb (unsigned short)

비트단위 OR 연산자

- 비트단위(bitwise) OR 연산자 활용
 - 피연산자의 특정한 위치의 비트를 1로 만들

예제 ↵

```
01 #include <iostream> ↵
02 using namespace std; ↵
03 ↵
04 int main() ↵
05 { ↵
06     unsigned short n = 0x6be9; ↵
07     unsigned short mask; ↵
08 ↵
09     mask = 0x0400; // 0x01 << 10, 오른쪽에서 11번째 비트 ↵
10     n = n | mask; ↵
11     cout.setf(ios::hex, ios::basefield); // 16진수로 출력 ↵
12     cout.setf(ios::showbase); // 16진수 앞에 0x를 출력하여 16진수임을 표현하도록함 ↵
13     cout << n << endl; ↵
14 ↵
15     return 0; ↵
16 }
```

0x6fe9 ↵

flag value	equivalent to
adjustfield	left right internal
basefield	dec oct hex
floatfield	scientific fixed

field	member constant	effect when set
independent flags	boolalpha	read/write bool elements as alphabetic strings (true and false).
	showbase	write integral values preceded by their corresponding numeric base prefix.
	showpoint	write floating-point values including always the decimal point.
	showpos	write non-negative numerical values preceded by a plus sign (+).
	skipws	skip leading whitespaces on certain input operations.
	unitbuf	flush output after each inserting operation.
	uppercase	write uppercase letters replacing lowercase letters in certain insertion operations.
numerical base (basefield)	dec	read/write integral values using decimal base format.
	hex	read/write integral values using hexadecimal base format.
	oct	read/write integral values using octal base format.
float format (floatfield)	fixed	write floating point values in fixed-point notation.
	scientific	write floating-point values in scientific notation.
adjustment (adjustfield)	internal	the output is padded to the field width by inserting fill characters at a specified internal point.
	left	the output is padded to the field width appending fill characters at the end.
	right	the output is padded to the field width by inserting fill characters at the beginning.

setf : format flag bit를 set하는 함수
unsetf : format flag bit를 unset(clear)하는 함수

basefield의 flag인 dec, oct, hex중에서 hex flag를 set함
즉 16진수를 출력하는 flat 를 set함

비트단위 AND 연산자

- 비트단위(bitwise) AND 연산자

- 피연산자: 정수
- 피연산자의 위치가 같은 비트들에 대해서 비트단위 AND 연산을 수행
- 예

m	0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1	0x6be9 (unsigned short)
n	1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 1	0xfa9b (unsigned short)
m & n	0 1 1 0 1 0 1 0 1 0 0 0 1 0 0 1	0x6a89 (unsigned short)

비트단위 AND 연산자

- 비트단위(bitwise) AND 연산자 활용
 - 피연산자의 특정한 위치의 비트를 1 인지 0인지를 검사 (bit masking)

예제 ↵

```
01. #include <iostream> ↵
02. using namespace std; ↵
03. ↵
04. int main() ↵
05. { ↵
06.     unsigned short n = 0x6be9; ↵
07.     unsigned short mask; ↵
08. ↵
09.     mask = 0x0100; // 0x01 << 8, 오른쪽에서 9번째 비트 ↵
10.     if( n & mask ) ↵
11.         cout << 1 << endl; ↵
12.     else ↵
13.         cout << 0 << endl; ↵
14. ↵
15.     return 0; ↵
16. }
```

1 ↵

비트단위 NOT 연산자

- 비트단위(bitwise) NOT 연산자
 - 피연산자: 정수
 - 피연산자의 위치가 같은 비트들에 대해서 비트단위 NOT 연산을 수행
 - 예

n	<div>0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1</div>	333 (unsigned short)
~n	<div>1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 0</div>	65202 (unsigned short)

비트단위 XOR 연산자

- 비트단위(bitwise) XOR 연산자
 - 피연산자: 정수
 - 피연산자의 위치가 같은 비트들에 대해서 비트단위 XOR 연산을 수행
 - 예

m 0 1 1 0 1 0 1 1 1 1 1 0 1 0 0 1 0x6be9 (unsigned short)

n 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0xfa9b (unsigned short)

m ^ n 1 0 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0x9172 (unsigned short)

비트단위 XOR 연산자

- 비트단위(bitwise) XOR 연산자 활용
 - 피연산자의 특정한 위치의 비트를 반대로(0->1, 1->0) 만드는데 사용

예제 ↵

```
01 #include <iostream> ↵
02 using namespace std; ↵
03 ↵
04 int main() ↵
05 { ↵
06     unsigned short n = 0x6be9; ↵
07     unsigned short mask; ↵
08 ↵
09     mask = 0x0200; // 0x01 << 9, 오른쪽에서 10번째 비트 ↵
10     n = n ^ mask; ↵
11     cout.setf(ios::hex, ios::basefield); // 16진수로 출력 ↵
12     cout.setf(ios::showbase); // 16진수 앞에 0x를 출력하여 16진수임을 표현하도록함 ↵
13     cout << n << endl; ↵
14 ↵
15     return 0; ↵
16 } ↵
```

0x69e9 ↵

비트단위 시프트 연산자

- 비트단위 시프트(bitwise shift) 연산자
 - 피연산자 : 정수형 데이터
 - 피연산자의 모든 비트를 왼쪽 혹은 오른쪽으로 이동시키는 연산자
 - 왼쪽 피연산자 : 이동 연산을 적용시킬 데이터
 - 오른쪽 피연산자 : 이동시킬 비트 수

비트단위 시프트 연산자	연산자의 계산
<<	왼쪽 피연산자의 모든 비트를 왼쪽으로 이동시킨다.
>>	왼쪽 피연산자의 모든 비트를 오른쪽으로 이동시킨다.

비트단위 시프트 연산자

- 왼쪽 시프트(left shift) 연산자

- $n \ll k$

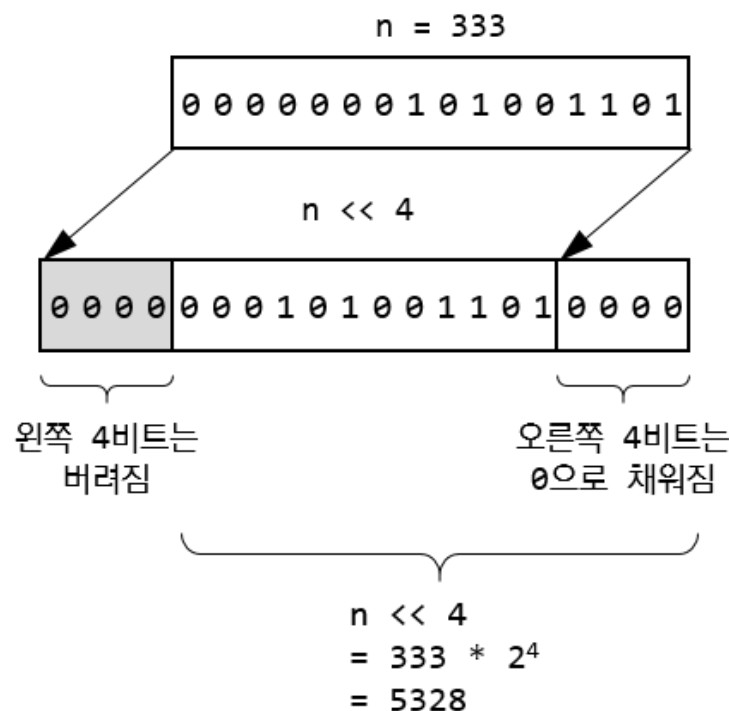
- 정수 n 을 k 비트만큼 왼쪽으로 이동시킨 값

- 왼쪽으로 이동시킬 때 가장 왼쪽 k 개 비트는 버려짐.

- 왼쪽으로 이동된 수만큼 비게 되는 오른쪽 k 개의 비트는 모두 0 비트로 채워짐

- 효과

- 결과값은 $n * 2^k$ 가 됨



비트단위 시프트 연산자

- 오른쪽 시프트(right shift) 연산자

- $n \gg k$

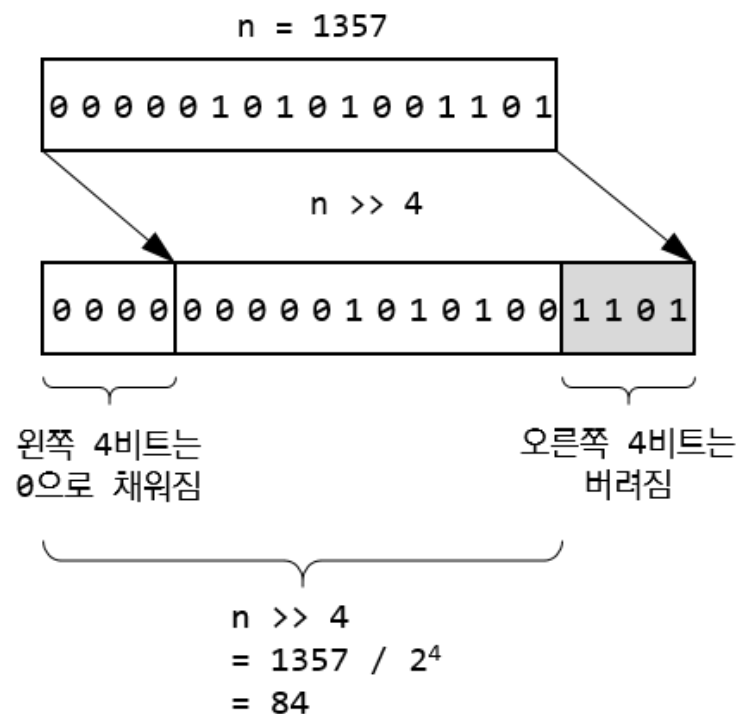
- 정수 n 을 k 비트만큼 오른쪽으로 이동시킨 값

- 오른쪽으로 이동시킬 때 가장 오른쪽 k 개 비트는 버려짐.

- 오른쪽으로 이동된 수만큼 비게 되는 왼쪽 k 개의 비트는 모두 MSB 비트로 채워짐

- 효과

- 결과값은 $n / 2^k$ 가 됨



연산자 사용시 주의할 점

- 비교 연산자 ==
 - 대입 연산자 = 로 잘못 사용하는 경우

예제 ↵

```
01  #include <iostream> ↵
02  using namespace std; ↵
03  ↵
04  int main() ↵
05  { ↵
06      int a = 1, b = 2; ↵
07  ↵
08      if(a = b) ↵
09          cout << a << " is equal to " << b << endl; ↵
10      else ↵
11          cout << a << " is not equal to " << b << endl; ↵
12  ↵
13      return 0; ↵
```

2 is equal to 2 ↵

연산자 사용시 주의할 점

- 비교 연산자

- 수식: $a \leq x \leq b$

$a \leq x \leq b$ = F // wrong

$(a \leq x) \ \&\& \ (x \leq b)$ // correct

예제 ↵

```
01 < #include <iostream> ↵
02 < using namespace std; ↵
03 < ↵
04 < int main() ↵
05 < { ↵
06 <     int a = -2; b = 0, c, x = -1; ↵
07 <     ↵
08 <     if(a <= x <= b)    // if( (a <= x) && (a <= b) ) ↵
09 <         c = 1; ↵
10 <     else ↵
11 <         c = 0; ↵
12 <     cout << c << endl; ↵
13 <     ↵
14 <     return 0; ↵
```

0 ↵

연산자 우선순위

	연산자	연산 내용	결합 순서
높음	::	영역 결정(scope resolution)	왼쪽 → 오른쪽
	() [] . -> ++ --	함수 호출 배열 첨자 구조체 원소 선택 포인터를 통한 구조체 원소 선택 후위 증감	왼쪽 → 오른쪽
	++ -- + - ! ~ (type) sizeof & * new delete	전위 증감 부호 (단항 연산자) 논리 NOT, 비트단위 NOT 형 변환 데이터 크기 (바이트 수) 주소 연산 역(간접)참조 연산 동적 메모리 할당 반환	오른쪽 → 왼쪽
	.* ->*	구조체 원소 역참조 포인터를 통한 구조체 원소 역참조	왼쪽 → 오른쪽
우선순위	* / %	곱셈, 나눗셈, 나머지	왼쪽 → 오른쪽
	+ -	덧셈, 뺄셈	
	<< >>	비트 단위 왼쪽 시프트, 오른쪽 시프트	
	> >= < <=	관계 연산 (크기 비교)	
	== !=	관계 연산 (같음, 다름)	
	&	비트 단위 AND	
	^	비트 단위 XOR	
		비트 단위 OR	
	&&	논리 AND	
		논리 OR	
낮음	?: = += -= *= /= %= <<= >>= &= ^= = throw	조건 연산 (삼항 연산자) 대입 연산 복합 대입 연산 (산술 연산) 복합 대입 연산 (비트단위 시프트 연산) 복합 대입 연산 (비트단위 AND, XOR, OR 연산) 예외 처리	오른쪽 → 왼쪽
	,	coma 연산	왼쪽 → 오른쪽

	연산자 기능 등에 따른 우선순위
높음	단항 연산자
	산술 연산
	비트 단위 시프트
	관계 연산
우선순위	비트 단위 연산 (시프트 제외)
	논리 연산
	조건 연산
	대입 연산
낮음	coma 연산

(주의사항)

- 조건연산자(?:)의 우선순위가 대입연산의 우선순위와 동일함
- C 언어에서는 조건연산자의 우선순위가 대입연산의 우선순위보다 높음

연산자 우선 순위 (예)

- (C언어) 다음 수식의 계산 결과는?

```
x = 0;  
x == 0 ? x += 1 : x += 2;  
printf("%d", x);
```

- 1) 0
- 2) 1
- 3) 2
- 4) 컴파일러 에러

```
x == 0 ? x += 1 : x += 2
```

C언어에서 위 수식에서 연산자의 우선 순위는 : == ?: +=
따라서 위 수식은 다음과 동일함

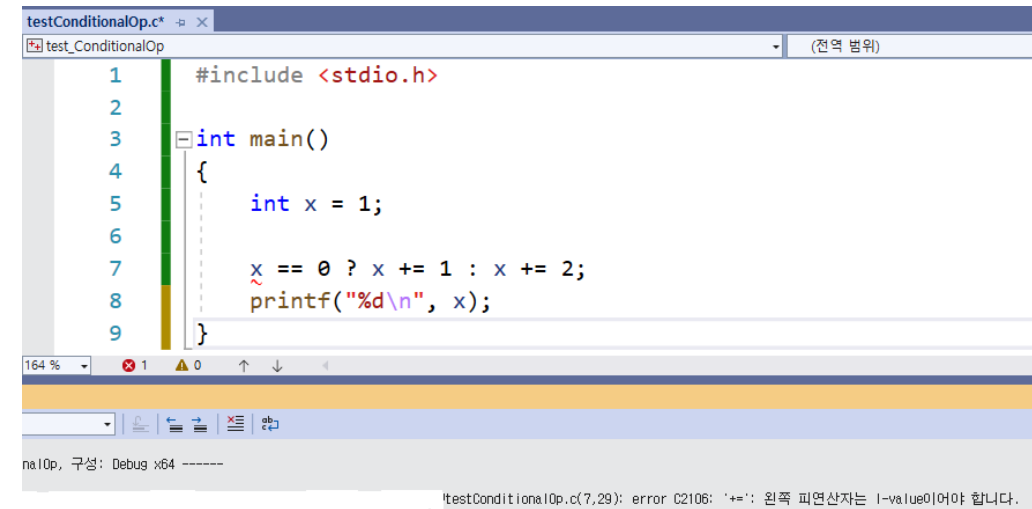
```
(x == 0 ? x += 1 : x) += 2
```

위 수식에서는 컴파일러 오류가 발생함

다음과 같은 수식을 기대하면 괄호를 사용하여 수식을 표현해야 함

```
(x == 0) ? (x += 1) : (x += 2)
```

Visual Studio, C 프로그램
컴파일러 오류



```
testConditionalOp.c*  
test_ConditionalOp (전역 범위)  
1 #include <stdio.h>  
2  
3 int main()  
4 {  
5     int x = 1;  
6  
7     x == 0 ? x += 1 : x += 2;  
8     printf("%d\\n", x);  
9 }
```

testConditionalOp.c(7,29): error C2106: '+=': 왼쪽 피연산자는 l-value이어야 합니다.

연산자 우선 순위 (예)

- (C++) 다음 수식의 계산 결과는?

```
x = 0;  
x == 0 ? x += 1 : x += 2;  
cout << x << endl;
```

- 1) 0
- 2) 1
- 3) 2
- 4) 컴파일러 에러

```
x == 0 ? x += 1 : x += 2
```

위 수식에서 연산자의 우선 순위는 : == (?: +=)

연산자 ?: 와 += 의 우선 순위는 동일함

연산자 ?: 와 += 는 오른쪽에서 왼쪽으로 결합하므로 위 수식은 다음과 같음

```
(x == 0) ? (x += 1) : (x += 2)
```

단축연산 (예)

- 다음 수식의 계산 결과는?

```
w = 0; x = 1; y = 2; z = 3;  
a = (++w > 0) && (++x > 1) && (y++ > 2) && (z++ > 3);  
cout << w << " " << x << " " << y << " " << z << endl;
```

False
실행 X

- 1) 0 1 2 3
- 2) 1 2 2 3
- 3) 1 2 2 4
- 4) 1 2 3 3
- 5) 1 2 3 4
- 6) 컴파일러 오류