작성자 : 김기희 (cupid072@naver.com)

연산자

- 학습목표
  - 연산자와 피연산자에 대해 설명할 수 있다.
  - 단항 연산자의 종류를 나열하고 연산자에 대해 설명할 수 있다.
  - 이항 연산자의 종류를 나열하고 연산자에 대해 설명할 수 있다.
  - 연산 기능에 따라 연산자를 구분하여 기술할 수 있다.
  - 최우선 연산자의 종류를 나열하고 연산자에 대해 설명할 수 있다.
  - 범정수 확장에 대해 설명할 수 있다.
  - 정수 변환 규칙에 대해 설명할 수 있다.
  - 일반적인 산술 변환에 대해 설명할 수 있다.
  - 부동소수점 연산의 오차 발생 원인과 해결 방법을 설명할 수 있다.

연산자와 피연산자

- 연산자(Operator)

  프로그램 상의 상수, 변수 등에 대하여 여러 조작을 하기 위한 기호

- 피연산자(Operand)

  연산자의 연산 적용 대상

- 피연산자의 수에 따른 연산자의 구분

  단항 연산자(Unary Operator), 이항 연산자(Binary Operator)

- 연산 기능에 따른 연산자의 구분

  참조연산, 산술연산, 논리연산, 비교연산, 비트연산, 대입 및 배정연산, 조건연산, 순차연산

작성자 : 김기희 (cupid072@naver.com)

연산자 우선순위

순위	명칭		연산자	연산방향
1	최우선 연산, 참조 연산		(expr) [] . ->	>
2	단항 연산		+ - ! ~ (type) sizeof ++ -- & *	<
3	이항 연산	승법 연산	* / %	>
4		가법 연산	+ -	
5		shift 연산	<< >>	
6		비교 연산	> >= < <=	
7		등가 연산	== !=	
8		bit 곱 연산	&	
9		bit 차 연산	^	
10		bit 합 연산	|	
11		논리곱 연산	&&	
12		논리합 연산	||	
13	조건 연산		? :	<
14	대입 연산		= += -= *= /= %= <<= >>= &= |= ^=	<
15	순차 연산		,	>

최우선, 참조 연산자

연산자	설명
(expr)	수식의 우선순위를 최우선으로 변경
[]	배열의 선언, 배열의 요소 지정
.	구조체 및 공용체 멤버 참조 (membershop operator)
->	구조체 및 공용체 간접 멤버 참조

수식을 감싸는 괄호는 연산의 우선순위를 최우선 순위로 높인다.

int a = 20 + 5 * 5; * 연산자가 우선순위가 높으므로 20 + 25가 되어 연산식의 결과는 45가 됨

※ 배열 연산자, 구조체 및 공용체 간접 멤버 참조 연산자는 배열과 구조체 공용체 강의 시간에 자세하게 다룰 예정임

단항 연산자

연산자	설명
+	Unary plus (부호 유지)
-	Unary minus (부호 반전)
!	논리부정 (NOT)
(type)	Cast(형변환) 연산자
&	주소(추출) 연산자
*	간접 참조 연산자
sizeof	변수, 상수 또는 데이터형의 크기(Byte)

- Unary - 연산자는 부호표시와 함께 피연산자의 부호를 반전시킨다.
- C 언어에서 0은 거짓이며 0이 아닌 값은 true로 해석된다.
- 논리 부정 연산자의 피연산자는 참 또는 거짓을 나타내는 식이어야 한다.
- sizeof 연산자는 피연산자에 할당된 메모리의 크기(Byte)를 구한다.

묵시적 형변환(promotion)과 명시적 형변환(cast)

- C 프로그래밍에서는 연산식에 따라 형변환이 발생한다.

- 형변환 시에는 다음과 같은 손실의 우려가 있다.
  부호의 손실 : signed 자료형을 unsigned 자료형으로 변환 시
  데이터의 손실 : 보다 큰 자료형을 작은 자료형으로 변환 시
  정밀도의 손실 : 실수 자료형을 정수형으로 변환하거나 큰 실수 형을 보다 작은 실수형으로 변환 시

- 형 변환에 있어서 위의 세 가지 손실의 우려가 존재하는 경우 명시적 형 변환을 사용하도록 한다.

```
long total = 295;                   // int형 상수를 long형 변수에 대입 (손실없음)
int count = 3;                      // int형 상수를 int형 변수에 대입 (손실없음)
double avg;
avg = total / count;                // 소수점 이하의 값을 구할 수 없음
avg = (double) total / count;       // double형 결과를 double형 변수에 대입
avg = (double)(total / count);      // long형 결과를 double형으로 형변환하여 대입
```

자료형의 크기

- 자료형의 크기는 다음과 같다.

  char < short < int < long < long long < float < double < long double

- 자료형의 바이트 수에 상관 없이 실수형은 모든 정수형보다 큰 자료형이다.

연산식에 있어서의 형 변환

- C 프로그램에서의 연산시의 형변환
  - ① 모든 피연산자에 대하여 범정수 확장을 한다. (정수 보다 작은 자료형은 정수형으로)
  - ② 범정수 확장은 원본 데이터 앞쪽에 바이트를 확장하여 4바이트 int형 데이터로 변환한다.
  - ③ 이때 앞 쪽에 채워지는 바이트의 비트는 다음의 값으로 채운다.
    - 정수확장 대상이 부호가 있는 정수(signed) 형인 경우 : 부호비트로 채움
    - 정수확장 대상이 부호가 없는 정수(unsigned) 형인 경우 : 0으로 채움
  - ④ 범정수 확장 후 두 피연산자가 동일한 int 형이면 연산을 수행하고 결과는 int형이 된다.
  - ⑤ 두 피연산자가 동일한 int형이며, 두 피연산자 중 하나는 부호가 있는 정수(signed)형이고 다른 하나가 부호가 없는 정수(unsigned)형인 경우 부호가 있는 정수(signed) 형이 부호가 없는 정수형(unsigned) 형으로 변환되어 연산을 수행한다.
  - ⑥ 범정수 확장 후 두 피연산자의 자료형이 다른 경우 작은 자료형을 보다 큰 자료형으로 형변환 한 후 연산을 수행한다.

정수확장

- 정수확장은 자료형의 앞 쪽에 바이트를 채워 특정 길이의 자료형으로 변환하는 것을 말한다.
  예) char 형을 int형으로 변환

- 정수확장시 앞쪽에 추가되는 바이트의 값은 다음의 값으로 채워진다.
  정수 확장 대상이 부호가 있는(signed) 정수인 경우 : 부호 비트로 채움
  정수 확장 대상이 부호가 없는(unsigned) 정수인 경우 : 0으로 채움

```
char ch = 65;
int i = ch;         00000000 00000000 00000000 01000001
char ch = -65;
int i = ch;         11111111 11111111 11111111 10111111
```

작은 자료형으로의 형변환

- 큰 자료형의 정수 값을 작은 자료형으로 변환 시 데이터를 구성하는 앞 쪽의 바이트를 버리고 뒤 쪽의 바이트만을 취한다.

  ☞ int형 65 값을 char 형으로 형변환 시

  ~~00000000 00000000 00000000~~ 01000001 > 65

  ☞ int형 257 값을 char형으로 형변환 시

  ~~00000000 00000000 00000001~~ 00000001 > 1

  ☞ int형 -129의 값을 char형으로 형변환 시

  ~~11111111 11111111 11111111~~ 01111111 > 127

- 실수형을 정수형으로 변환 시 소수점 이하는 버려지고 정수부만 취한다.

```
double pi = 0.314e+1;      // 3.14의 지수표기
int p = pi;                // 3
```

연산식에 있어서의 형 변환

- 대입 연산자의 형 변환

  Rv 값이 Lv 자료형에 맞추어 대입된다.

```
double pi = 3.14;
int p = pi;              // 소수점 이하는 버려지고 정수부만 저장 (묵시적 형변환)
int p = (int) pi;        // 프로그래머에 의한 명시적 형변환
```

- 위의 묵시적 형 변환이 컴파일 에러가 발생하는 것은 아님.
- 묵시적 형변환을 프로그래머가 의도 한 것인가? 아니면 실수 한 것인가? 알 수 없음
- 명시적 형변환 시 프로그래머에 의해 의도적으로 형변환 된 것임이 명확해짐

이항 연산자

연산자	설명
*	곱하기
/	몫
%	나머지
+	더하기
-	빼기
> >= <= <	비교연산
== !=	등가연산

- 서로 다른 우선순위의 연산자가 두 개 이상 연속해서 사용되는 연산식을 복합 연산식이라고 한다.
- 복합 연산식은 괄호를 사용하여 연산의 우선순위를 명확히 할 것을 권장한다.

```
int a = (3 * 5) + 10;          // 연산의 우선 순위가 * 연산자가 높더라도 괄호를 사용할 것을 권장
```

- 비교 연산자의 피연산자는 산술식(수치 값을 나타내는 식)이 되도록 한다.
- 논리식(논리 값을 나타내는 식)에 대한 평가에는 등가연산자를 사용하도록 한다.

비교 연산자의 형 변환

- 다음 두 코드의 결과를 예상해 보세요.

```
char x = -1;
unsigned char y = 1;

if (x > y) {
    printf("true");
}
else {
    printf("false");
}
```

```
int x = -1;
unsigned int y = 1;

if (x > y) {
    printf("true");
}
else {
    printf("false");
}
```

비교 연산자의 형 변환

```
char x = -1;
unsigned char y = 1;

if (x > y) {
    printf("true");
}
else {
    printf("false");
}
```

- x와 y 두 변수 모두 범정수 확장에 의해 int 형으로 형변환 된다.

  x의 형변환 값 : -1

  y의 형변환 값: 1
- x의 값이 y의 값 보다 작다.
- 그러므로 "false" 가 출력된다.

비교 연산자의 형 변환

```
int x = -1;
unsigned int y = 1;

if (x > y) {
    printf("true");
}
else {
    printf("false");
}
```

- 변수 x와 y 모두 int 형이므로 범정수 확장은 발생하지 않는다.
- 그러나 unsigned int와 signed int형 간의 연산이 되므로 signed int형이 unsigned int형으로 형변환 된다.

  x 의 값 : 1

  y의 unsigned int형 변환 값 : 2147483646

- x의 값이 y의 값 보다 작다.
- 그러므로 "true" 출력

비트 연산자

연산자	설명
~	1의 보수 (One's complement) 연산자
|	비트 단위 논리합(Bitwise OR)
&	비트 단위 논리곱(Bitwise AND)
^	비트 단위 배타적 논리합(Bitwise Exclusive OR)
<<	Left shift 연산자
>>	Right shift 연산자

- ~ (1의 보수) 전체 비트를 1로 만들기 위해 더해야 하는 값을 구한다.

  ~128 : 128의 비트 전체를 1로 만들기 위해 더해야 하는 값을 구한다.

```
128의 비트 구성  00000000 00000000 00000000 10000000
          +    11111111 11111111 11111111 01111111      > -129
               11111111 11111111 11111111 11111111
```

비트 연산자

- | : 비트단위 논리곱 (두 비트를 곱했을 때 0이 아니면 1)

  3 | 15

  3의 비트 구성 00000011

  15의 비트 구성 00001111

  00000011

- & : 비트단위 논리합 (두 비트를 더했을 때 0이 아니면 1)

  3 | 15

  3의 비트 구성 00000011

  15의 비트 구성 00001111

  00001111

비트 연산자

- ^ : 비트단위 배타적 논리합 (두 비트를 더했을 때 1이어야 1)

  3 ^ 15

  3의 비트 구성 00000011

  15의 비트 구성 00001111

  00001100

비트 연산자

- >> (Right shift)연산자 : 비트를 오른쪽으로 이동
  ① 좌측 피연산자의 비트를 우측 피연산자의 수 만큼 우측으로 이동시킨다.
  ② 좌측에 새로 들어오는 비트는 다음의 값으로 채워진다.
    - 부호가 있는(signed) 정수형 : 부호 비트
    - 부호가 없는(unsigned) 정수형 : 0
  ③ 우측에 밀려 나가는 비트는 버려진다.

- << (Left shift)연산자 : 비트를 왼쪽으로 이동
  ① 좌측 피연산자의 비트를 우측 피연산자의 수 만큼 좌측으로 이동시킨다.
  ② 좌측에 밀려 나가는 비트는 버려진다.
  ③ 우측에 새로 들어오는 비트는 부호가 있는(signed) 정수, 부호가 없는(unsigned) 정수에 상관없이 0으로 채워진다.

비트 연산자

- >> 연산자

```
int a = 15;          00001111        양수 값에 대한 Right shift
a >>= 2;             0000001111      // 15 / 2 / 2 = 3
int a = -15          11110001        음수 값에 대한 Right shift
a >> 2;              1111110001      -4
```

- << 연산자

```
int a = 3;           00000011        양수 값에 대한 Left shift
a <<= 2;           0000001100        // 3 * 2 * 2 = 12
int a = -3;          11111101        음수 값에 대한 Left shift
a <<= 2;           1111110100        // -3 * 2 * 2 = -12
```

비트 연산자

- shift 연산자 사용시 주의사항
- shift 연산은 다음의 규칙에 의해 수행된다.

  a = a << 3 -> a << (3 % a의 데이터를 구성하는 비트 수)

  a = a << 32 -> a << 0

  a = a << 33 -> a << 1
- 그러므로 우측 피연산자의 값은 좌측 피연산자의 비트 수 미만의 값이 되도록 한다.
- 되도록이면 Right shift 연산은 부호가 없는 정수형에 대해서만 수행하도록 할 것을 권장함.
- shift 연산자 사용 시 정수확장(범정수확장 포함)을 고려해야 한다.

```
unsigned char c = 255;
c <<= 8;    // c의 값은?
```

변수 c의 비트 구성	11111111
범정수 확장에 의해 int형으로 형 변환	00000000 00000000 00000000 11111111
shift 연산 수행	00000000 00000000 11111111 00000000
char형으로 축소하여 저장	~~00000000 00000000 11111111~~ 00000000

논리 연산자

연산자	설명
&&	논리곱 (Logical AND)
||	논리합 (Logical OR)

- 참을 1, 거짓을 0이라고 할 때
  - && (논리곱) : 두 피연산자를 곱하였을 때 0이 아니면 참
  - || (논리합) : 두 피연산자를 더했을 때 0이 아니면 참

- 논리 연산자의 피연산자 평가
  - && (논리곱) : 좌측 피연산자가 거짓이면 우측 피연산자를 평가하지 않는다.
  - || (논리합) : 좌측 피연산자가 참이면 우측 피연산자를 평가하지 않는다.

- 논리 연산자의 피연산자는 논리식(논리 값을 나타내는 식)으로 할 것을 권장함.

증감 연산자

연산자	설명
++	증가(Increment) 연산자
--	감소(Decrement) 연산자

- 전위 연산자 (연산자를 우선 기술)
  - ++a : 값을 1 증가 시킨 후 참조
  - --a : 값을 1 감소 시킨 후 참조
- 후위 연산자
  - a++ : 값을 먼저 참조한 후 1 증가
  - a-- : 값을 먼저 참조한 후 1 감소

컴파일러에 따라 ++, -- 연산식의 결과는 상수이다.
아래의 연산식은 gcc에서는 허용되지 않지만 Visual Studio에서는 허용된다.

```
++(++a);
```

※ 하나의 시퀀스에서 한 변수에 대한 부작용(side effect)은 오직 한 번만 허용하도록 한다.
위의 경우 하나의 시퀀스에서 변수 a에 대하여 두 번의 데이터 변경이 수행되므로 NG

조건(삼항) 연산자

연산자	설명
?:	조건 ? 참일 때 : 거짓일 때

```
int age = 19;
int adult = (age >= 19) ? 1 : 0;          // age가 19 이상이면 adult 값을 1, 19 미만이면 adult 값을 0으로
.....
if (adult == 1) {
      .....
}
```

- Visual Studio의 경우 아래와 같은 사용도 허용

```
(age >= 19) ? adult = 1 : adult = 0;          // 호환성을 위해 사용하지 말 것을 권장
```

대입 및 배정 연산자

연산자	설명
=	대입 연산자
+= ...	좌측의 피연산자와 우측의 피연산자에 대하여 연산을 수행한 후 좌측의 피연산자에 대입

- 우측 피연산식의 값(RV)을 좌측의 변수(LV)에 대입한다.
- 좌측의 피연산자는 오직 변수만 기술할 수 있다.
- 우측의 피연산자는 변수, 리터럴, 연산식, 함수 등 모든 식이 기술될 수 있다.

- int a = 10;
- int b = 10;
- a += b; // RV에 변수(변수식)을 사용
- a += 10; // RV에 리터럴(상수식)을 사용
- a += function(); // RV에 함수호출(함수식)을 사용
- a += (10 * 2) + 3; // RV에 연산식을 사용