원리를 알면 IT가 맛있다
IT COOKBOOK

IT CookBook, C++ 하이킹 객체지향과 만나는 여행

[강의교안 이용 안내]

C++ 하이킹

객체지향과 만나는 여행

Chapter 02. 자료형과 연산자

한빛아카데미 Hanbit Academy, Inc.

목차

학습목표

- 자료형에 대한 개념을 이해한다.
- C++에서 사용하는 다양한 자료형에 대해 학습한다.
- 산술, 관계, 논리, 증감, 대입 연산자의 사용법을 익힌다.
- 비트 단위 연산자와 조건 연산자, sizeof 연산자, 형변환을 위한 캐스트 연산자를 학습한다.

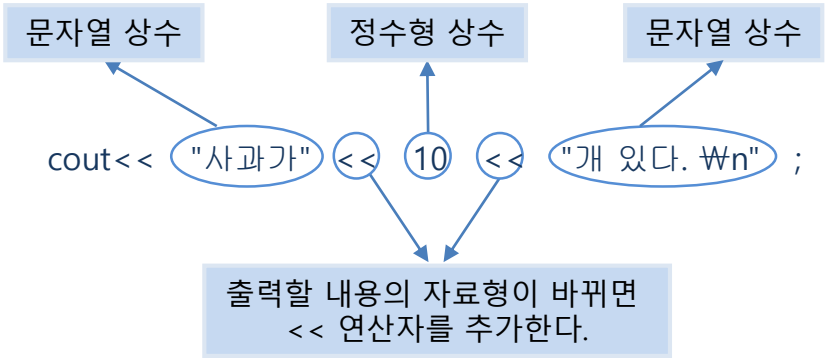
01. 자료형의 이해

자료형의 개념

- 프로그램을 작성하기 위한 자료의 형태를 정해 두었는데, 이를 '자료형'이라 한다.

상수와 변수의 개념

- 정수형 상수 : 변하지 않는 고유의 값을 '상수(constant)'라 한다.



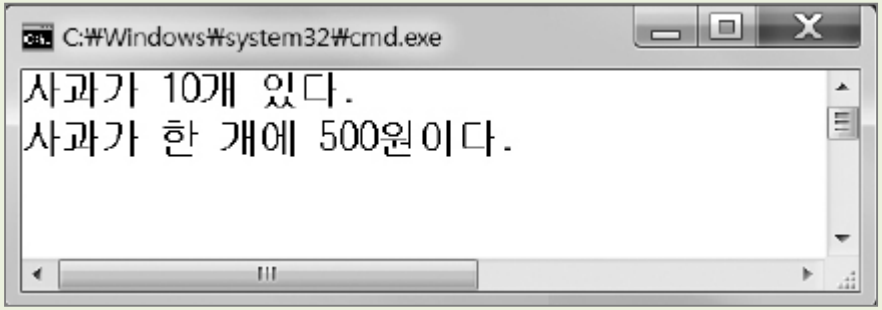
- 정수형 변수 : 변수는 프로그램 수행 중에 상수(값)를 저장할 수 있는 기억공간(그릇)으로 다양한 상수(값)을 저장할 수 있다. 변수 선언문은 다음과 같이 자료형과 변수명으로 구성된다.

변수 선언 기본 형식

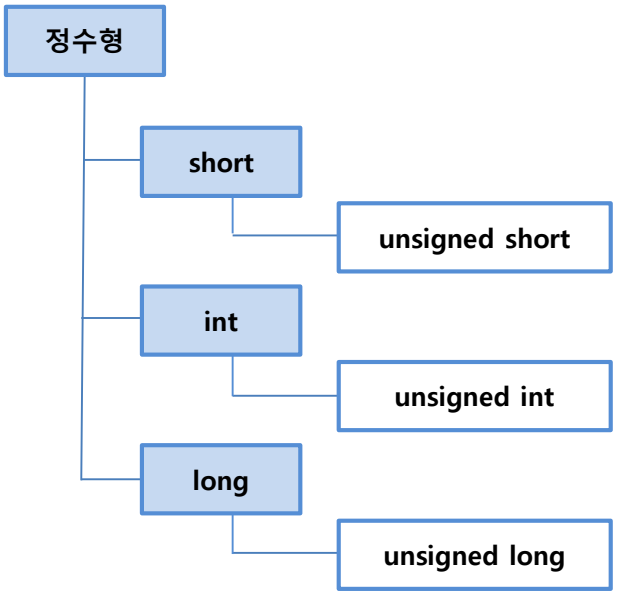
```
자료형 변수명
```

예제 2-1. 정수형 상수 출력하기(02_01.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 cout<<"사과가 "<< 10 <<"개 있다.₩n";
06 cout<<"사과가 한 개에 "<< 500 <<"원이다.₩n";
07 }
```



01. 자료형의 이해



[그림 2-1] 정수형 변수를 선언할 때 사용하는 자료형의 종류

- 식별자 정의 규칙

① 영문자(A~Z, a~z)와 숫자(0~9), 밑줄문자(_)의 조합으로 만들어진다.
② 첫 글자는 반드시 영문자나 _로 시작해야 한다. 숫자로 시작해서는 안 된다.
③ 식별자는 철자(스펠링)가 같다고 해도 대소문자를 구분하기 때문에 조심해야 한다.
④ C++에서 사용되는 예약어는 식별자로 사용할 수 없다.
⑤ 식별자는 가급적이면 자기 역할에 맞는 이름을 부여한다.

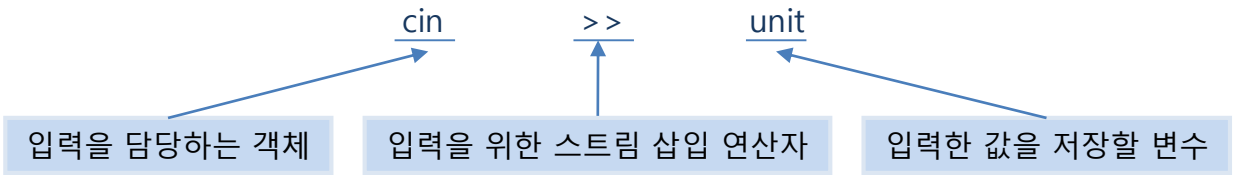
예제 2-2. 정수형 변수 사용하기(02_02.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int unit; // 변수 선언
06 int count;
07 int total;
08 unit=500; // 한 개에 500원짜리
09 count=5; // 5개를 구입한
10 total=unit*count; // 총 금액 구하기
11 cout<<" 총 금액 :" << total <<"₩n";
12 }
```

01. 자료형의 이해

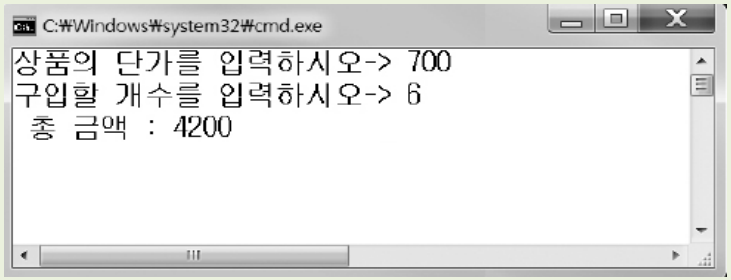
입력 담당 객체 cin

- cin 객체는 >> 연산자 다음에 기술된 변수에 키보드에서 입력받은 값을 저장한다.

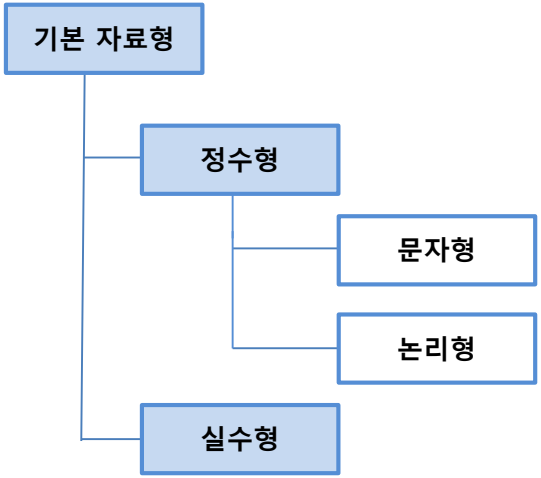


예제 2-3. cin을 이용해서 변수 입력받기(02_03.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int unit, count, total; // 변수 선언
06 cout<<"상품의 단가를 입력하시오-> ";
07 cin>>unit;
08 cout<<"구입할 개수를 입력하시오-> ";
09 cin>>count;
10 total=unit*count; // 키보드에서 입력받은 데이터로 총금액 구하기
11 cout<<" 총 금액 : " << total <<"₩n";
12 }
```



02. 자료형의 종류



[그림 2-2] C++ 기본 자료형의 종류

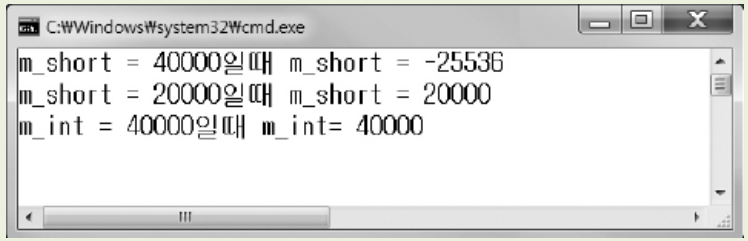
- **정수형 :** 크게 short, int, long으로 나뉜다.

[표 2-1] 정수형 자료형의 종류

자료형	크기	저장할 수 있는 값의 범위	
short	2바이트(16비트)	-2 ~2 -1	-32768 ~ 32767
unsigned short	2바이트(16비트)	0 ~ 2^{16} -1	0 ~ 65535
int	2바이트(16비트)	-2^{31} ~ 2^{31} -1	-2147483648 ~ 2147483647
unsigned int	2바이트(16비트)	0 ~ 2^{32} -1	0 ~ 4294967295
long	2바이트(16비트)	-2^{31} ~ 2^{31} -1	-2147483648 ~ 2147483647
unsigned long	2바이트(16비트)	0 ~ 2^{32} -1	0 ~ 4294967295

예제 2-4. 정수형 오버플로우 에러가 발생하는 예 살펴보기(02_04.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 short m_short = 40000;
06 cout << "m_short = 40000일때 m_short = " << m_short <<"₩n";
07 m_short = 20000;
08 cout << "m_short = 20000일때 m_short = " << m_short <<"₩n";
09 int m_int = 40000;
10 cout << "m_int = 40000일때 m_int= " << m_int <<"₩n";
11 }
```

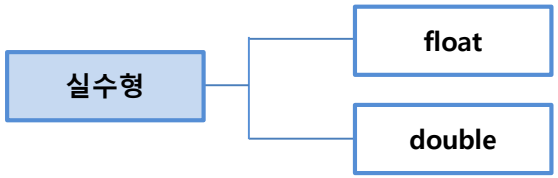


02. 자료형의 종류

■ **실수형 :** 소수점이 있는 수를 뜻한다.

[표 2-2] 실수형 상수의 예

상수의 종류	예	의미
소수형	1234.5 또는 0.0000987	가장 일반적으로 사용하는 실수형 데이터
지수형	1.2345E3 또는 0.987E-5	영문자 E를 기준으로 앞에는 가수부, 뒤에는 지수부를 기술함



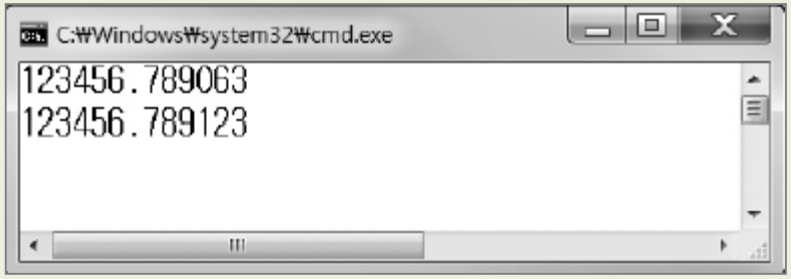
[그림 2-3] 실수형의 종류

[표 2-3] 실수형의 종류

유형	크기	유효범위
float	4바이트	$\pm 3.4\times10^{-38} \sim \pm 3.4\times10^{3}$
double	8바이트	$\pm 1.7\times10^{-308} \sim \pm 1.7\times10^{30}$

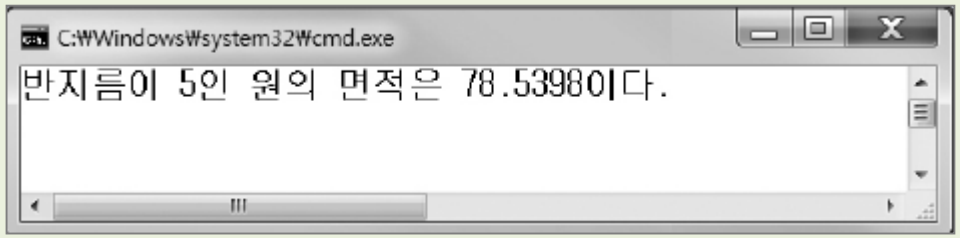
예제 2-5. 실수형 오버플로우 에러가 발생하는 예 살펴보기(02_05.cpp)


```
01 #include <iostream>
02 #include <iomanip> // setiosflags 조작기 사용을 위해 포함한 헤더파일
03 using namespace std;
04
05 void main()
06 {
07 float a=123456.789123;
08 double b=123456.789123;
09
10 // 실수를 소수점 형태로 출력하겠다는 의미
11 cout<<setiosflags(ios::fixed);
12 cout<<a<<endl;
13 cout<<b<<endl;
14 }
```



예제 2-6. 원의 면적을 실수형으로 구하기(02_06.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 double pi=3.141592;
06 int r=5;
07 double area;
08
09 area=r*r*pi;
10
11 cout << "반지름이 "<<r<<"인 원의 면적은 "<<area<<"이다.₩n";
12 }
```



02. 자료형의 종류

문자형

- 문자형 상수는 한 개의 한영문자, 숫자, 특수문자 등을 표현하는 자료형으로, 작은따옴표(' ')로 묶어서 표현한다.

대문자

'A'	'B'	'C'	'D'	'E'	'F'	...	'X'	'Y'	'Z'
65	66	67	68	69	70	...	88	89	90

소문자

'a'	'b'	'c'	'd'	'e'	'f'	...	'x'	'y'	'z'
97	98	99	100	101	102	...	120	121	122

숫자

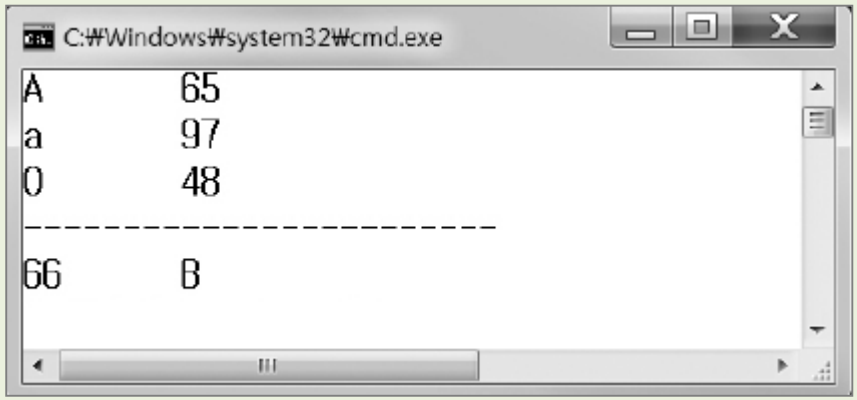
'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
48	49	50	51	52	53	54	55	56	57

[그림 2-4] 아스키코드 값

예제 2-7. 문자형 상수와 아스키코드의 관계 알아보기(02_07.cpp)

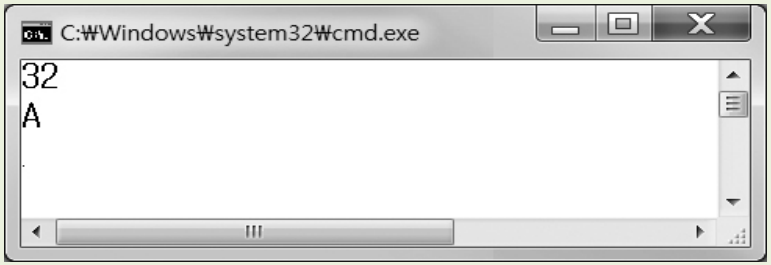
원리를 알면 IT가 맛있다
IT COOKBOOK

```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 cout << 'A' <<" ";
06 cout << (int) 'A' <<"₩n";
07 cout << 'a' <<" ";
08 cout << (int) 'a' <<"₩n";
09 cout << '0' <<" ";
10 cout << (int) '0' <<"₩n";
11 cout << "----------------------₩n";
12 cout << 'A'+1 <<"₩t";
13 cout << (char) ('A'+1) <<"₩n";
14 }
```



예제 2-8. 소문자를 대문자로 변경하기(02_08.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 // 소문자와 대문자는 아스키코드 값이 32 차이가 남
06 cout << 'a' - 'A' <<"₩n";
07 // 소문자에서 32를 빼면 대문자가 구해짐
08 cout << (char) ('a' -32) <<"₩n";
09 }
```



02. 자료형의 종류

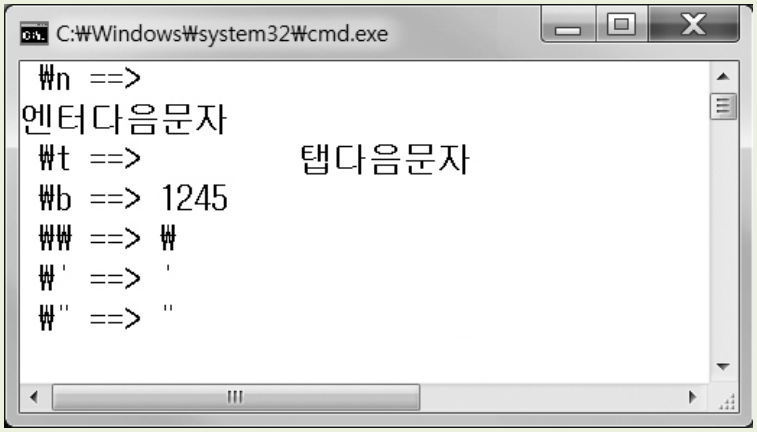
■ 확장 특수문자

[표 2-4] 확장 특수문자의 종류

확장 특수문자	의미
₩n	<Enter> 키의 기능을 하며 줄을 바꾼다(New Line).
₩t	수평 탭으로 일정한 간격을 띄운다(horizontal tab).
₩b	백스페이스 기능으로 뒤로 한 칸 후진한다(backspace).
₩r	동일한 줄의 맨 앞으로 커서만 옮긴다(carriage return).
₩f	출력 용지를 한 페이지 넘긴다(form feed).
₩a	경고음을 낸다(alert).
₩ \	\ 문자를 출력한다(back slash).
₩'	' 문자를 출력한다(single quote).
₩"	" 문자를 출력한다(double quote).
₩0	널(Null) 문자다.

예제 2-9. 확장 특수문자 출력하기(02_09.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 cout<<" ₩₩n ==> " <<"₩n"<<"엔터다음문자"<<endl;
06 cout<<" ₩₩t ==> " <<"₩t"<<"탭다음문자"<<endl;
07 cout<<" ₩₩b ==> " <<"123"<<"₩b"<<"45"<<endl;
08 cout<<" ₩₩₩₩ ==> " << "₩₩"<<endl;
09 cout<<" ₩₩₩' ==> " << "₩'"<<endl;
10 cout<<" ₩₩₩" ==> " << "₩""<<endl;
11 }
```



02. 자료형의 종류

문자형의 종류

- char형은 단일 문자 상수를 저장하기 위해 사용한다.

[표 2-5] char형의 크기와 유효 범위

유형	크기	유효범위
char	1바이트(8비트)	-128 ~ 127
unsigned char	1바이트(8비트)	0 ~ 255

문자열형

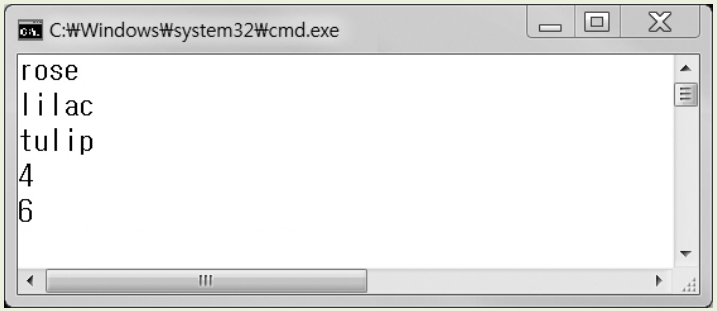
- 문자열(string) 상수는 일련된 문자들의 집합으로 구성되고 반드시 큰따옴표(" ")로 둘러싸야 한다.
- 문자열로 사용하려면 끝을 나타내는 널(NULL) 문자, 즉 ₩0을 맨 마지막 문자로 추가해 주어야 한다.

논리형

- true와 false를 이용해 참과 거짓을 나타낼 수 있는 논리형(bool)이 있다.

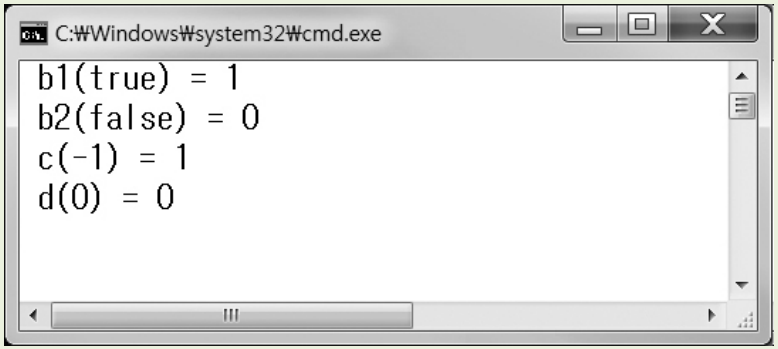
예제 2-10. 문자열 저장하기(02_10.cpp)


```
01 #include <iostream>
02 #include <cstring> // strlen() 함수를 사용하기 위한 헤더파일
03 using namespace std;
04 void main()
05 {
06 char flowers1[5] = "rose";
07 char flowers2[6] = {'l', 'i', 'l', 'a', 'c'};
08 char flowers3[] = "tulip";
09
10 cout << flowers1 <<"₩n";
11 cout << flowers2 <<"₩n";
12 cout << flowers3 <<"₩n";
13
14 cout << strlen(flowers1) <<"₩n"; // 문자열의 길이
15 cout << sizeof(flowers3) <<"₩n"; // 배열의 크기
16 }
```



예제 2-11. 논리형이 저장되는 형태 살펴보기(02_11.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 bool b1 = true, b2 = false;
06 cout <<" b1(true) = " << b1 <<"₩n";
07 cout <<" b2(false) = " << b2 <<"₩n";
08 bool c = -1, d = 0;
09 cout <<" c(-1) = " << c <<"₩n";
10 cout <<" d(0) = " << d <<"₩n";
11 }
```



03. 기본 연산자

산술 연산자

- 산술 연산자(arithmetic operation)는 피연산자에 대한 덧셈, 뺄셈, 곱셈, 나눗셈을 하는 연산자다.

[표 2-6] 산술 연산자의 종류

구분	연산자	의미	수학적 표현	C++ 표현	C++ 결과
단항 연산자	+	양수	+3	+3	3
	-	음수	-2(-2)	-2(-2)	4
이항 연산자	+	덧셈	3+2	3+2	5
	-	뺄셈	10-2	10-2	8
	*	곱셈	xy 또는 x×y	2*4	8
	/	나눗셈	x/y 또는 x÷y	8/2	4
	%	나머지	x mod y	9%2	1

산술 연산자 사용시 주의사항

① 피연산자의 자료형에 의해서 결과값이 달라질 수 있다.

② 이항 연산자 중 나머지를 구하는 %는 정수형 데이터만 피연산자로 취할 수 있다.

03. 기본 연산자

산술 연산자의 우선순위

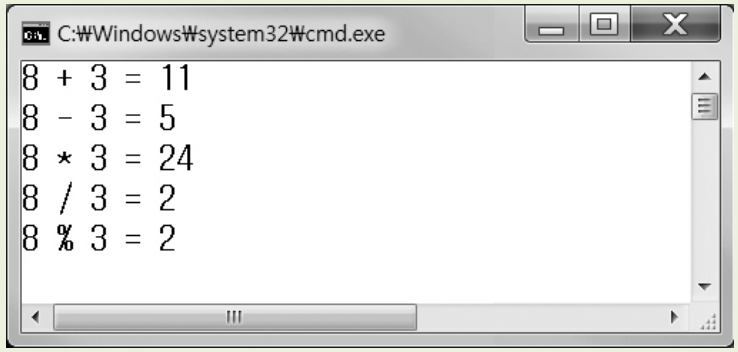
① 괄호 안에 있는 연산자가 가장 먼저 계산된다. 괄호 여러 개가 중첩되어 있을 경우 가장 안쪽의 괄호가 먼저 계산된다.

② *(곱셈), /(몫을 구하는 나눗셈), %(나머지를 구하는 나눗셈) 연산자가 다음으로 계산된다. 수식 안에 *, /, % 등 연산자가 여러 개 있을 경우 왼쪽에서 오른쪽으로 연산이 실행된다. 이 3가지 연산자 사이의 우선순위는 같다.

③ +(덧셈), -(뺄셈) 연산자가 그 다음으로 계산된다. 수식 안에 +, - 연산자가 여러 개있을 경우 왼쪽에서 오른쪽으로 연산이 실행된다. 이 2가지 연산자 사이의 우선순위는 같다.

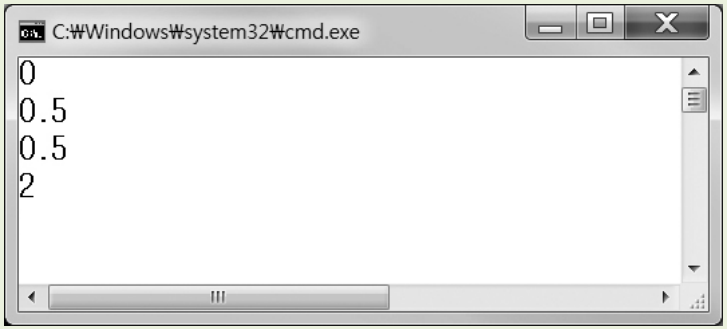
예제 2-12. 산술 연산자 사용하기(02_12.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int a = 8, b = 3;
06 cout<<a<<" + "<<b<<" = "<<a+b<<"₩n";
07 cout<<a<<" - "<<b<<" = "<<a-b<<"₩n";
08 cout<<a<<" * "<<b<<" = "<<a*b<<"₩n";
09 cout<<a<<" / "<<b<<" = "<<a/b<<"₩n";
10 cout<<a<<" % "<<b<<" = "<<a%b<<"₩n";
11 }
```



예제 2-13. 나누기 연산자 사용하기(02_13.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 cout<<2/4<<"₩n";
06 cout<<2/4.0<<"₩n";
07 cout<<2.0/4.0<<"₩n";
08 cout<<2%4<<"₩n";
09 // cout<<2.0%4.0<<"₩n";
10 }
```



03. 기본 연산자

관계 연산자

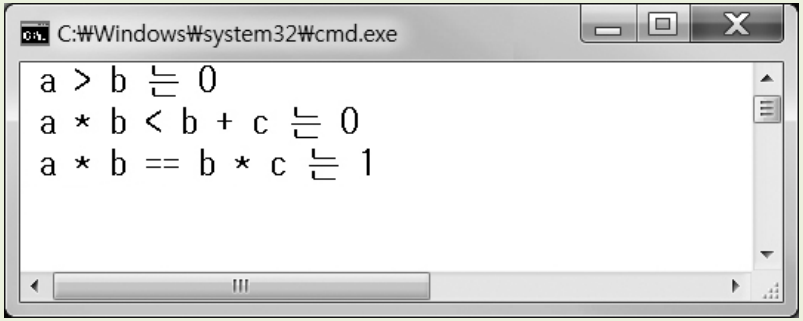
- 관계 연산자(relational operator)는 숫자 2개를 비교하는 연산자로 연산의 결과는 항상 참(true) 또는 거짓(false)이 된다.

[표 2-7] 관계 연산자의 종류

구분	연산자	의미	수학적 표현	C++ 표현	C++ 결과
항등 연산자	==	좌측이 우측과 같다.	=	2 == 4	false
	!=	좌측이 우측과 같지 않다.	≠	2 != 4	true
비교 연산자	>	좌측이 우측보다 크다.	>	3 > 2	true
	>=	좌측이 우측보다 크거나 같다.	≥	2 >= 3	false
	<	좌측이 우측보다 작다.	<	4 < 5	true
	<=	좌측이 우측보다 작거나 같다.	≤	4 <= 4	true

예제 2-14. 관계 연산자 사용하기(02_14.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int a = 3, b = 5, c = 3;
06 bool istrue;
07 istrue = a > b;
08 cout<<" a > b 는 " << istrue << "₩n";
09
10 istrue = a * b < b + c ;
11 cout<<" a * b < b + c 는 " << istrue << "₩n";
12
13 istrue = a * b == b * c ;
14 cout<<" a * b == b * c 는 " << istrue << "₩n";
15 }
```



03. 기본 연산자

논리 연산자

- 논리 연산자(logical operator)는 일련의 조건을 모두 만족하는 경우나 일부만 만족하는 경우를 구별하는 데 사용한다.

[표 2-8] 논리 연산자의 종류

종류	설명
&&	논리곱 연산자(logical AND operator)
||	논리합 연산자(logical OR operator)
!	단항 논리부정 연산자(logical NOT operator)

[논리 연산자 &&와 ||의 진리표]

값		&&	||
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

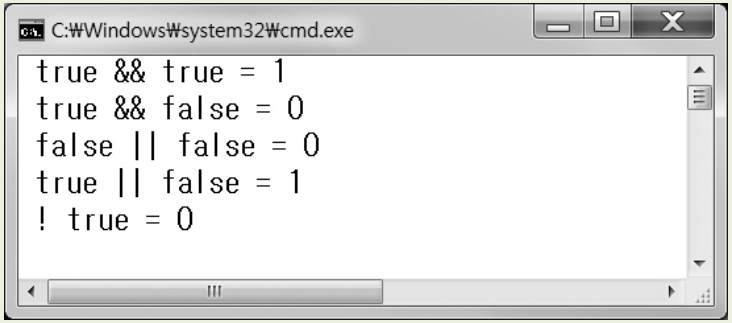
[논리 연산자 !의 진리표]

값	!
0	1
1	0

[그림 2-5] 논리 연산에 대한 진리표

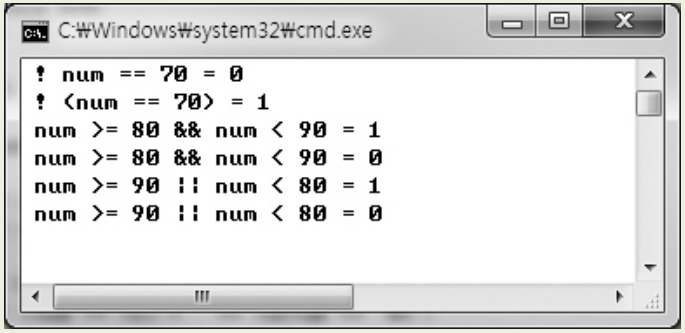
예제 2-15. 논리값에 논리 연산자 사용하기(02_15.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 bool istrue;
06
07 istrue = true && true;
08 cout<< " true && true = " << istrue << "₩n";
09
10 istrue = true && false;
11 cout<< " true && false = " << istrue << "₩n";
12
13 istrue = false || false;
14 cout<< " false || false = " << istrue << "₩n";
15
16 istrue = true || false;
17 cout<< " true || false = " << istrue << "₩n";
18
19 istrue = ! true;
20 cout<< " ! true = " << istrue << "₩n";
21 }
```



예제 2-16. 관계 연산자와 논리 연산자 혼용해서 사용하기(02_16.cpp)


```
01 include <iostream>
02 using namespace std;
03 void main()
04 {
05 int num=85;
06
07 bool istrue;
08
09 istrue = ! num == 70;
10 cout<< " ! num == 70 = " << istrue << "₩n";
11
12 istrue = ! (num == 70);
13 cout<< " ! (num == 70) = " << istrue << "₩n";
14
15 istrue = num >= 80 && num < 90;
16 cout<< " num >= 80 && num < 90 = " << istrue << "₩n";
17
18 num=60;
19
20 istrue = num >= 80 && num < 90;
21 cout<< " num >= 80 && num < 90 = " << istrue << "₩n";
22
23 istrue = num >= 90 || num < 80;
24 cout<< " num >= 90 || num < 80 = " << istrue << "₩n";
25
26 num=85;
27 istrue = num >= 90 || num < 80;
28 cout<< " num >= 90 || num < 80 = " << istrue << "₩n";
29 }
```



03. 기본 연산자

증감 연산자

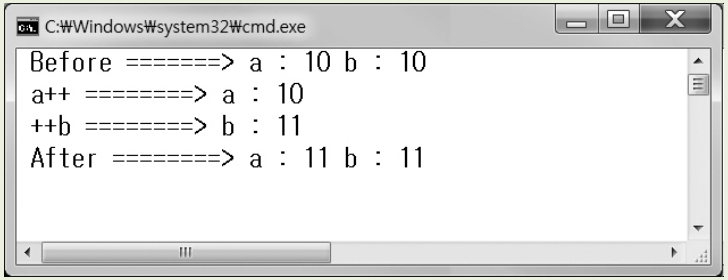
- 프로그램을 작성하다 보면 변숫값을 1만큼 증가시키나 1만큼 감소시켜야 하는 경우가 자주 발생한다. 이때 이용하는 연산자가 바로 증감 연산자다.

[표 2-9] 증감 연산자의 종류

연산자	형태	명칭	의미
++	++a	전위 증가 연산자	연산 전에 a값 증가
	a++	후위 증가 연산자	연산 후에 a값 증가
--	--a	전위 감소 연산자	연산 전에 a값 감소
	A--	후위 감소 연산자	연산 후에 a값 증가

예제 2-17. 증가 연산자 사용하기(02_17.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int a=10, b=10;
06 cout<<" Before ========> a : "<<a<<" b : "<<b<<"₩n";
07 cout<<" a++ =========> a : "<<a++<<"₩n";
08 cout<<" ++b =========> b : "<<++b<<"₩n";
09 cout<<" After =========> a : "<<a<<" b : "<<b<<"₩n";
10 }
```



대입 연산자

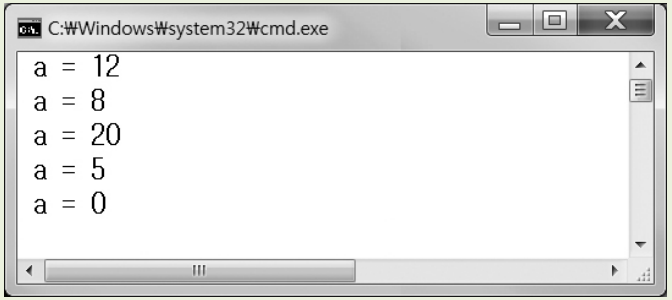
- 대입 연산자는 일반적으로 대입 연산자 왼쪽에 기술한 변수에 대입 연산자 오른쪽에 기술한 식의 결과를 저장하려고 사용한다.

[표 2-10] 대입 연산자의 종류

연산자	사용 예	같은 의미의 수식
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y
&=	x&=y	x=x&y
|=	x|=y	x=x|y
^=	x^=y	x=x^y
>>=	x>>=y	x=x>>y
<<=	x<<=y	x=x<<y

예제 2-18. 다양한 형태의 대입 연산자 사용하기(02_18.cpp)


```
01 #include<iostream>
02 using namespace std;
03 void main()
04 {
05 int a , b = 2;
06
07 a = 10; a += b; cout << " a = " << a<< "₩n";
08 a = 10; a -= b; cout << " a = " << a<< "₩n";
09 a = 10; a *= b; cout << " a = " << a<< "₩n";
10 a = 10; a /= b; cout << " a = " << a << "₩n";
11 a = 10; a %= b; cout << " a = " << a << "₩n";
12 }
```



04 비트 단위 연산자

- 비트 단위 연산자는 비트 연산자(bitwise operator)의 비트를 직접 조작하는 데 사용한다.

[표 2-11] 비트 연산자의 종류

연산자	명칭	의미
&	비트 논리곱(AND)	두 피연산자의 대응되는 두 비트가 모두 1이면 결과 비트는 1이 된다.
|	비트 논리합(OR)	두 피연산자의 대응되는 두 비트 중 적어도 한 비트가 1이면 결과 비트는 1이 된다.
^	비트 배타적 논리합(XOR)	두 피연산자의 대응되는 두 비트 중 한 비트가 1이면(두 비트가 서로 다른 비트일 때) 결과 비트는 1이 된다.
<<	왼쪽 이동(left shift)	두 번째 피연산자의 지정 개수만큼 첫 번째 피연산자의 비트들을 왼쪽으로 이동(시프트)한다. 오른쪽은 0으로 채운다.
>>	오른쪽 이동(right shift)	두 번째 피연산자의 지정 개수만큼 첫 번째 피연산자의 비트들을 오른쪽으로 이동한다. 왼쪽에 값을 채우는 방법은 시스템에 따라 다르다.
~	1의 보수(one's complement)	모든 0비트는 1이 되고 모든 1비트는 0이 된다.

04 비트 단위 연산자

비트 AND, 비트 OR, 비트 XOR, 비트 NOT 연산자

[표 2-12] 비트 AND, 비트 OR, 비트 XOR의 결과

대응 피연산자 비트		비트 AND	비트 OR	비트 XOR
X	Y	(X&Y)	(X|Y)	(X^Y)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

[표 2-13] 1의 보수(NOT) 연산 결과

피연산자 비트 X	비트 NOT(~X)
0	1
1	0

04 비트 단위 연산자

비트 연산 과정

- Short x = 10, y = 6;

- 비트 연산을 위해 먼저 이 10진수를 2진수로 변환해야 하는데, 10을 2진수로 변환하면

  $10 = 8 + 2 = 2^3 + 2^1$이 된다. 그리고 short형이므로 다음과 같이 2바이트(16비트)에서 2^3 과 2^1자리에 1을 표시하고 나머지는 0으로 채워진다.

10

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

- 그리고 각 6을 2진수로 변환하면 $6 = 4 + 2 = 2^2 + 2^1$이 되고 2^2 자리에 1, 2^1 자리에 1을 표시하고 나머지는 0으로 채워진다.

6

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

04 비트 단위 연산자

비트 AND 연산

- 2진수로 변환된 10과 6에 비트 AND 연산(&)을 수행하면 다음과 같이 대응되는 비트가 둘 다 1이면 1이고, 그렇지 않으면 0이다. 2의 1승 자리만 1이므로 $2^1 = 2$가 된다.

10	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
&	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

비트 OR 연산

- 2진수로 변환된 10과 6에 비트 OR 연산(|)을 수행하면 다음과 같이 대응되는 비트가 둘 중에 하나 이상 1이면 1이고, 둘다 0이면 0이다. 2의 3승 자리, 2의 2승 자리, 2의 1승 자리가 1이므로 $2^3 + 2^2 + 2^1 = 8 + 4 + 2 = 14$가 된다.

10	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
|	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

04 비트 단위 연산자

비트 XOR 연산

- 2진수로 변환된 10과 6에 비트 exclusive-OR 연산(^)을 수행하면 다음과 같이 대응되는 비트가 서로 다르면 1이고 둘다 0이거나 둘다 1이면 0이다. 2의 3승 자리, 2의 2승 자리가 1이므로 $2^3 + 2^2 = 8 + 4 = 12$가 된다.

10	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
^	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

04 비트 단위 연산자

비트 NOT 연산

- 2진수로 변환된 10에 비트 NOT 연산(~)을 수행하면 다음과 같다.

10	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
~	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

- 위와 같이 좌측 부호 비트가 1로 채워졌으므로 음수가 되고 음수는 2의 보수 형태로 저장된다. 2의 보수는 1의 보수 +1이고 1의 보수는 지금과 같이 비트 단위로, 1은 0으로 0은1로 변경하는 비트 NOT 연산과 같다. 그럼 결과값으로 주어진 2진수가 10진수로 얼마를 의미하는지 알아보자.

❶ ~의 연산 결과

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

❷ ~연산 결과에 대한 1의 보수

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

❸ 결과에 대한 2의 보수 = 1의 보수 + 1

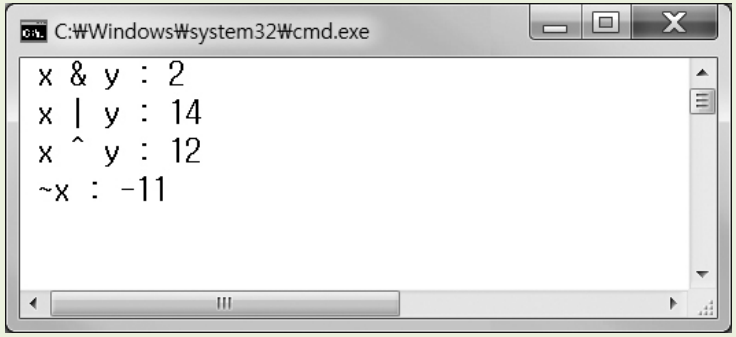
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

- 이 값을 10진수로 바꾸면 11이고 이 값에 - 기호를 붙인 -11이 결과로 주어진 2진수의 값이다. 즉, 다음의 2진수는 10진수로 -11이다.

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

예제 2-19. 비트 연산자 사용하기(02_19.cpp)


```
01 #include<iostream>
02 using namespace std;
03 void main()
04 {
05 short x = 10, y = 6 ;
06 cout << " x & y : " << (x & y) << "₩n";
07 cout << " x | y : " << (x | y) << "₩n";
08 cout << " x ^ y : " << (x ^ y) << "₩n";
09 cout << " ~x : " << (~x) << "₩n";
10 }
```



04 비트 단위 연산자

시프트 연산자

- 시프트 연산자는 2진수를 비트 단위로 이동하는 연산자다. 시프트 연산자를 사용하면 왼쪽에 오는 피연산자를 2진수로 변환하여 오른쪽으로 오는 피연산자 개수만큼 이동한다.

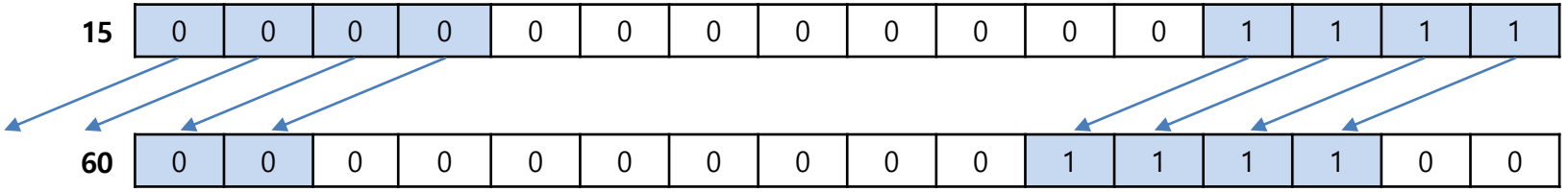
[표 2-14] 시프트 연산자의 종류

연산자	명칭	사용 예	의미
<<	왼쪽 시프트 연산자	a << 1	a의 값을 왼쪽으로 1비트만큼 이동시킨다.
>>	오른쪽 시프트 연산자	a >> 1	a의 값을 오른쪽으로 1비트만큼 이동시킨다.

04 비트 단위 연산자

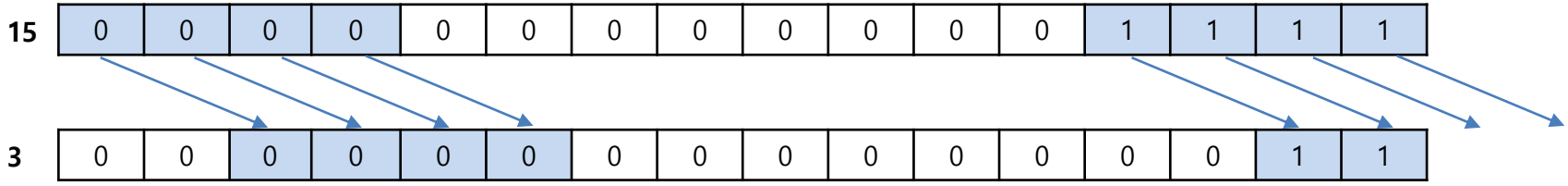
왼쪽 시프트 연산자

- 변수 x에 저장된 15에 왼쪽 시프트 연산(<<)을 x << 2와 같이 적용하면 다음과 같이 변수 x에 저장되어 있는 15를 2비트 왼쪽으로 이동시킨다. 그리고 왼쪽 시프트 연산자는 오른쪽의 빈 곳을 0으로 채우는데, 왼쪽에 저장할 공간이 없으면 버려진다. 그러므로 결과는 60이 된다.



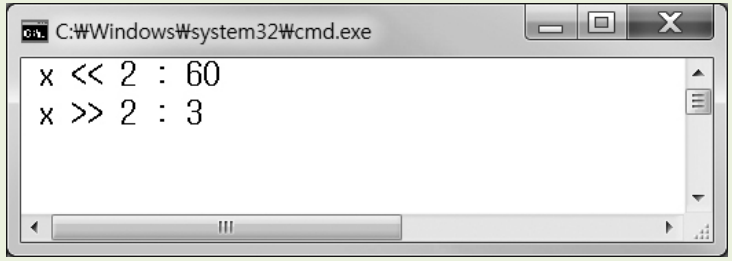
오른쪽 시프트 연산자

- 변수 x에 저장된 15에 오른쪽 시프트 연산(>>)을 x >> 2와 같이 적용하면 다음 그림과 같이 변수 x에 저장되어 있는 15를 2비트 오른쪽으로 이동시킨다. 그리고 왼쪽 빈 곳은 첫번째 비트가 양수면 0으로, 음수면 1로 채우는데, 오른쪽에 저장할 공간이 없으면 버려진다.



예제 2-20. 시프트 연산자 사용하기(02_20.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 short int x = 15;
06
07 cout << "x << 2 : " << (x << 2) << "₩n";
08 cout << "x >> 2 : " << (x >> 2) << "₩n";
09 }
```



05 기타 연산자

조건 연산자

- 조건 연산자(conditional operator)는 조건식의 결과인 true와 false에 따라 문장을 골라서 수행하는 연산자로서, 피연산자 3개를 갖는 삼항 연산자(ternary operator)다. 조건식의 결과가 참이면 '식1'을, 거짓이면 '식2'를 수행한다.

조건 연산자 기본 형식

```
조건식 ? 식1 : 식2;
```

sizeof 연산자

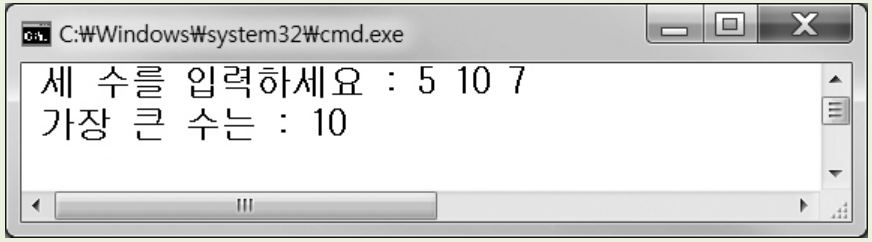
- sizeof 연산자는 변수명이나 자료형 또는 상숫값에 적용하여 메모리에 차지하는 바이트수를 구하기 위한 연산자다. 그리고 연산 결과로 되돌리는 값은 정수 형태다.

sizeof 연산자 기본 형식

```
sizeof(자료형/변수/상수)
```

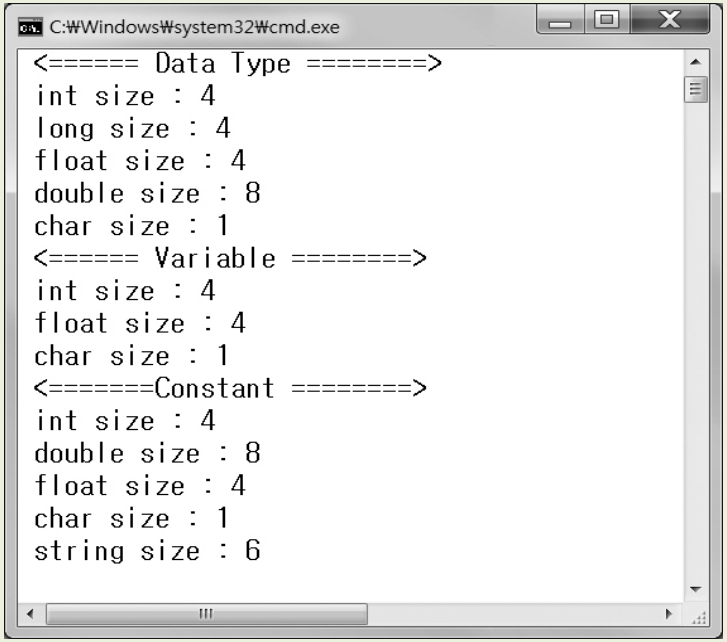
예제 2-21. 조건 연산자를 이용해서 최댓값 구하기(02_21.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int a , b , c;
06 int max;
07
08 cout << " 세 수를 입력하세요 : ";
09 cin >> a >> b >> c;
10
11 max = (a > b) ? a : b;
12 max = (max > c) ? max : c;
13
14 cout << " 가장 큰 수는 : " << max <<"₩n";
15 }
```



예제 2-22. sizeof 연산자로 메모리 크기 구하기(02_22.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int a=10;
06 float b=3.5f;
07 char c='A';
08 cout<<"₩n <======= Data Type ========>";
09 cout<<"₩n int size : "<<sizeof(int);
10 cout<<"₩n long size : "<<sizeof(long);
11 cout<<"₩n float size : "<<sizeof(float);
12 cout<<"₩n double size : "<<sizeof(double);
13 cout<<"₩n char size : "<<sizeof(char);
14
15 cout<<"₩n <======= Variable ========>";
16 cout<<"₩n int size : "<<sizeof(a);
17 cout<<"₩n float size : "<<sizeof(b);
18 cout<<"₩n char size : "<<sizeof(c);
19
20 cout<<"₩n <======== Constant ========>";
21 cout<<"₩n int size : "<<sizeof(23);
22 cout<<"₩n double size : "<<sizeof(3.5);
23 cout<<"₩n float size : "<<sizeof(3.5f);
24 cout<<"₩n char size : "<<sizeof('A');
25 cout<<"₩n string size : "<<sizeof("Apple")<<"₩n";
26 }
```



05 기타 연산자

형변환과 캐스트 연산자

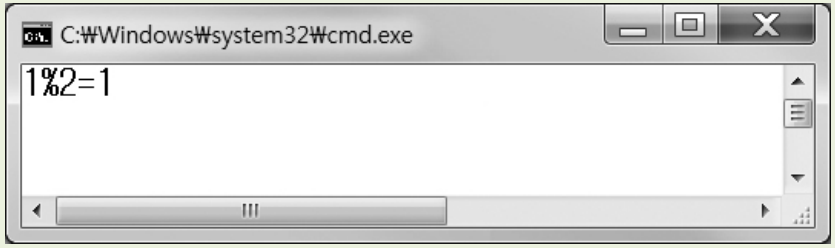
- 대입 연사자를 두고 자료형이 서로 다를 때 대입 연산자 오른쪽의 자료형이 왼쪽에 오는 변수의 자료형에 맞추어서 자료형을 변경하는 것을 형변환(type Conversion)이라 한다.
  ① 광역화 형변환 : 기본 자료형(int, long, float, double)에 대한 형변환 시에 작은 자료형에서 큰 자료형으로 값이 변환하는 것을 의미한다. 컴파일러에 의해서 자동으로 수행되는데, 이렇게 컴파일러에 의해서 형변환이 일어나는 것을 '묵시적 형변환'이라고 한다.
  ② 협소화 형변환 : 큰 자료형에서 작은 자료형으로 값이 변환하는 것을 의미한다. 협소화 형변환은 피연산자의 값이 상실될 우려가 있으므로 컴파일러가 경고 메시지를 발생시킨다. 경고를 받지 않으려면 프로그래머가 캐스트 연산자를 사용하여 '명시적 형변환'을 해야 한다.

- ( ) 안에 프로그래머가 식의 자료형에서 바꾸고자 하는 자료형을 기입하면 식의 자료형이 변환된다. 식에는 변수, 상수, 수식이 모두 올 수 있다. 명시적 형변환은 경고를 무시하고 강제로 형변환하는 것이므로 '강제 형변환'이라 칭하기도 한다.

캐스트 연산자 기본 형식	캐스트 연산자 사용 예
(자료형) 식	int num01 = (int) 3.5; double num02 = (double) 5;

예제 2-23. % 연산자를 사용하기 위해 강제 형변환하기(02_23.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 double a=1.0;
06 double b=2.0;
07 // int res01 = a % b;
08 int res01 = (int)a % (int)b;
09
10 cout<<a<<"%"<<b<<"="<<res01<<endl;
11 }
```



예제 2-24. 원하는 결과를 얻기 위해 강제 형변환하기(02_24.cpp)


```
01 #include <iostream>
02 using namespace std;
03 void main()
04 {
05 int c=1;
06 int d=2;
07 double res02;
08 res02= c / d;
09 cout<<c<<"/"<<d<<"="<<res02<<endl;
10 res02 = (double)c/(double)d;
11 cout<<c<<"/"<<d<<"="<<res02<<endl;
12 }
```