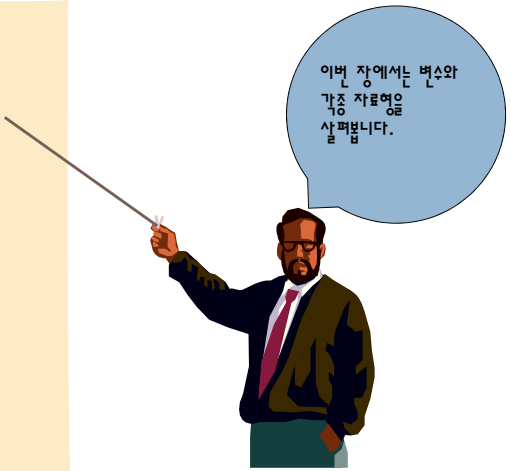
이번 장에서 학습할 내용

* 변수와 상수의 개념 이해
* 자료형
* 정수형
* 실수형
* 문자형
* 기호 상수 사용
* 오버플로우와 언더플로우 이해



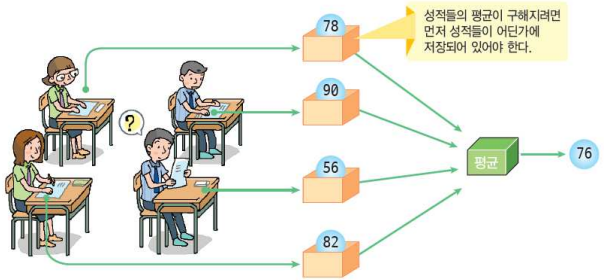
변수

Q) 변수(variable)이란 무엇인가?

A) 프로그램에서 일시적으로 데이터를 저장하는 공간

Q) 변수는 왜 필요한가?

A) 데이터가 입력되면 어딘가에 저장해야만 다음에 사용할 수 있다.



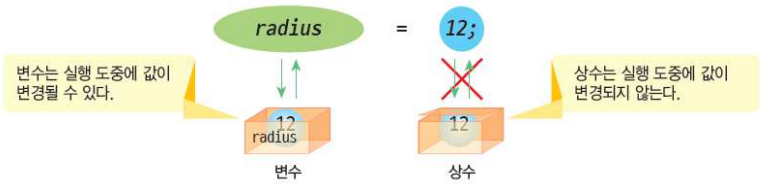
변수를 사용하는 이유

변수를 사용하지 않는 코드	변수를 사용하는 코드

`// 크기가 100×200인 사각형의 면적`
`area = 100 * 200;`
	`// 크기가 width×height인 사각형의 면적`
	`width = 100;`
	`height = 200;`
	`area = width * height;`

변수와 상수

- 변수(variable): 저장된 값의 변경이 가능한 공간
- 상수(constant): 저장된 값의 변경이 불가능한 공간
  - (예) 3.14, 100, 'A', "Hello World!"



예제: 변수와 상수

```
/* 원의 면적을 계산하는 프로그램 */
#include <stdio.h>
int main(void)
{
        float radius;          // 원의 반지름
        float area;            // 원의 면적

        printf("원의 면적을 입력하시요:");
        scanf("%f", &radius);

        area = 3.141592 * radius * radius;
        printf("원의 면적: %f \n", area);

        return 0;
}
```

변수

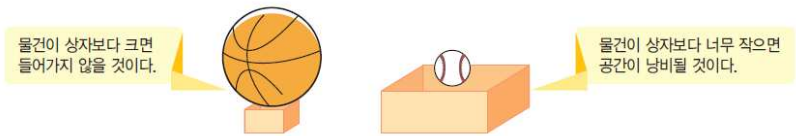
상수

자료형

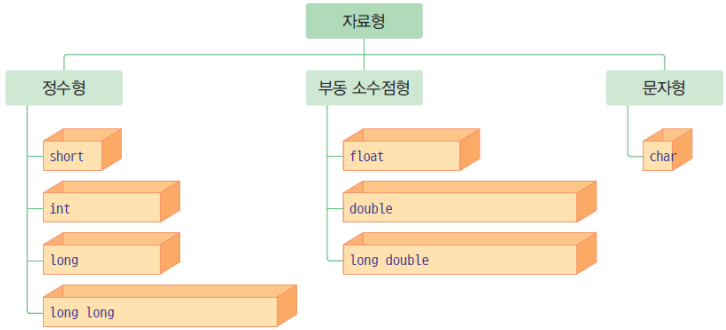
- 자료형(data type): 데이터의 타입(종류)
  - (예) short, int, long: 정수형 데이터(100)
  - (예) double, float:실수형 데이터(3.141592)
  - (예) char: 문자형 데이터('A', 'a', '한')

다양한 자료형이 필요한 이유

- 상자에 물건을 저장하는 것과 같다.

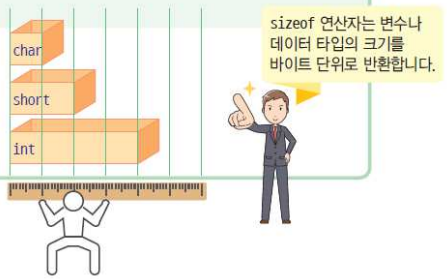


자료형



자료형의 크기

```
Syntax: sizeof()

예  sizeof(x)        // 변수
    sizeof(10)       // 값
    sizeof(int)      // 자료형
    sizeof(double)   // 자료형
```



예제: 자료형의 크기

```
#include <stdio.h>

int main(void)
{
        int x;
        printf("변수 x의 크기: %d\n", sizeof(x));
        printf("char형의 크기: %d\n", sizeof(char));
        printf("int형의 크기: %d\n", sizeof(int));
        printf("short형의 크기: %d\n", sizeof(short));
        printf("long형의 크기: %d\n", sizeof(long));
        printf("long long형의 크기: %d\n", sizeof(long long));
        printf("float형의 크기: %d\n", sizeof(float));
        printf("double형의 크기: %d\n", sizeof(double));
        return 0;
}
```

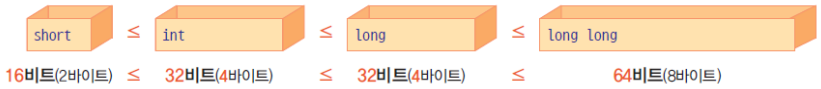

실행 결과

실행결과

```
변수 x의      크기: 4
char형의      크기: 1
int형의       크기: 4
short형의     크기: 2
long형의      크기: 4
long long형의    크기: 8
float형의     크기: 4
double형의    크기: 8
```

정수형

- short, int, long, long long



정수형

자료형			비트	범위
정수형	short	부호있는 정수	16비트	-32768~32767
	int		32비트	-2147483648~2147483647
	long			-2147483648~2147483647
	long long		64비트	-9,223,372,036,854,775,808 ~9,223,372,036,854,775,807
	unsigned short	부호없는 정수	16비트	0~65535
	unsigned int		32비트	0~4294967295
	unsigned long			0~4294967295
	unsigned long long		64비트	0~18,446,744,073,709,551,615

정수형 선언의 예

- short grade; // short형의 변수를 생성한다.
- int count; // int형의 변수를 생성한다.
- long distance; // distance형의 변수를 생성한다.

정수형의 범위

약 -21억에서 +21억

- int형

-2^{31}, ..., -2, -1, 0, 1, 2, ..., 2^{31} -1
(-2147483648 ~ +2147483647)

- short형

-2^{15}, ..., -2, -1, 0, 1, 2, ..., 2^{15} -1
(-32768 ~ +32767)

- long형
  - 보통 int형과 같음

예제

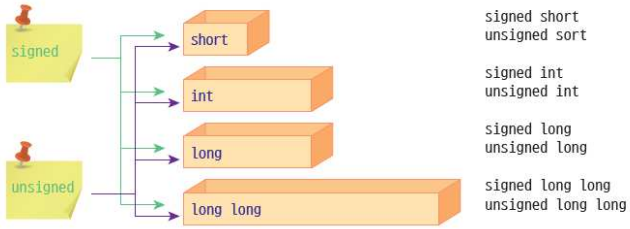
```c
/* 정수형 자료형의 크기를 계산하는 프로그램*/
#include <stdio.h>

int main(void)
{
	short year = 0;			// 0으로 초기화한다.
	int sale = 0;			// 0으로 초기화한다.
	long total_sale = 0;		// 0으로 초기화한다.
	long long large_value;

	year = 10;			// 약 3만2천을 넘지 않도록 주의
	sale = 200000000;		// 약 21억을 넘지 않도록 주의
	total_sale = year * sale;	// 약 21억을 넘지 않도록 주의

	printf("total_sale = %d \n", total_sale);

	return 0;
}
```

실행결과

```
total_sale = 2000000000
```

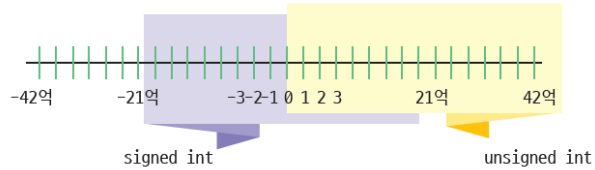
signed, unsigned 수식자

- unsigned
  - 음수가 아닌 값만을 나타냄을 의미
  - unsigned int
- signed
  - 부호를 가지는 값을 나타냄을 의미
  - 흔히 생략



unsigned int

$0, 1, 2, \ldots, 2^{32} - 1$
(0 ~ +4294967295)

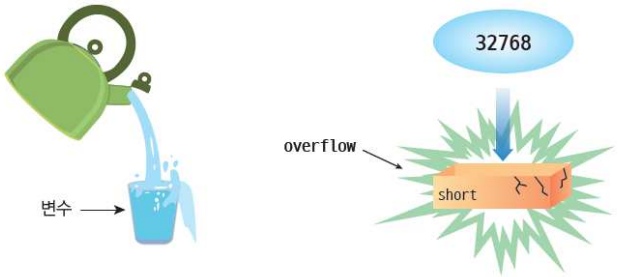


unsigned 수식자

- unsigned int speed;// 부호없는 int형
- unsigned distance; // unsigned int distance와 같다.
- unsigned short players; // 부호없는 short형
- unsigned long seconds; // 부호없는 long형

오버플로우

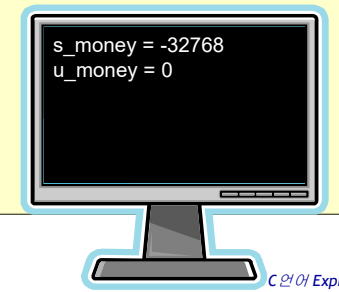
- 오버플로우(overflow): 변수가 나타낼 수 있는 범위를 넘는 숫자를 저장하려고 할 때 발생



오버플로우

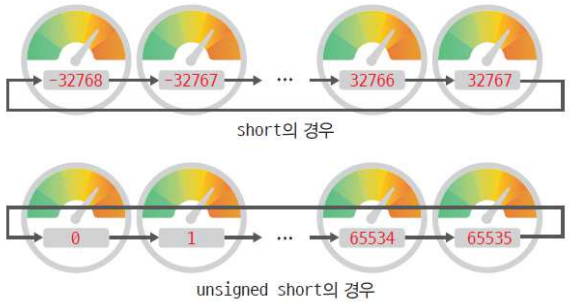
```
#include <stdio.h>
#include <limits.h>

int main(void)
{
        short s_money = SHRT_MAX;              // 최대값으로 초기화한다. 32767
        unsigned short u_money = USHRT_MAX;  // 최대값으로 초기화한다. 65535

        s_money = s_money + 1;
        printf("s_money = %d", s_money);

        u_money = u_money + 1;
        printf("u_money = %d", u_money);
        return 0;
}
```

오버플로우 발생!!



오버플로우

- 규칙성이 있다.
  - 수도 계량기나 자동차의 주행거리계와 비슷하게 동작



정수 상수

- 숫자를 적으면 기본적으로 int형이 된다.
  - sum = 123; // 123은 int형
- 상수의 자료형을 명시하려면 다음과 같이 한다.
  - sum = 123L; // 123은 long형

접미사	자료형	예
u 또는 U	unsigned int	123u 또는 123U
l 또는 L	long	123l 또는 123L
ul 또는 UL	unsigned long	123ul 또는 123UL

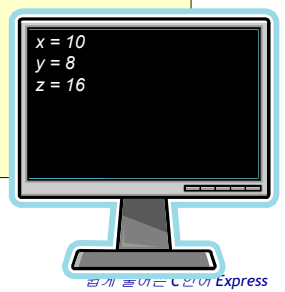
10진법, 8진법, 16진법

- 8진법
  - $012_8 = 1\times 8^1 + 2\times 8^0 = 10$
- 16진법
  - $0xA_{16} = 10\times 16^0 = 10$

10진수	8진수	16진수
0	00	0x0
1	01	0x1
2	02	0x2
3	03	0x3
4	04	0x4
5	05	0x5
6	06	0x6
7	07	0x7
8	010	0x8
9	011	0x9
10	012	0xa
11	013	0xb
12	014	0xc
13	015	0xd
14	016	0xe
15	017	0xf
16	020	0x10
17	021	0x11
18	022	0x12

예제

```c
/* 정수 상수 프로그램*/
#include <stdio.h>

int main(void)
{
    int x = 10;   // 10은 10진수이고 int형이고 값은 십진수로 10이다.
    int y = 010;  // 010은 8진수이고 int형이고 값은 십진수로 8이다.
    int z = 0x10; // 010은 16진수이고 int형이고 값은 십진수로 16이다.

    printf("x = %d", x);
    printf("y = %d", y);
    printf("z = %d", z);

    return 0;
}
```



기호 상수

- *기호 상수(symbolic constant):* 기호를 이용하여 상수를 표현한 것
- (예)
  - area = 3.141592 * radius * radius;
  - area = PI * radius * radius;
  - income = salary * 0.15 * salary;
  - income = salary - TAX_RATE * salary;
- 기호 상수의 장점
  - 가독성이 높아진다.
  - 값을 쉽게 변경할 수 있다.

기호 상수의 장점

```c
#include <stdio.h>


int main(void)
{
  ...
  won1 = 1050 * dollar1;
  won2 = 1050 * dollar2;
  ...
}
```

리터럴 상수를 사용하는 경우:
등장하는 모든 곳을 수정하여야 한다.

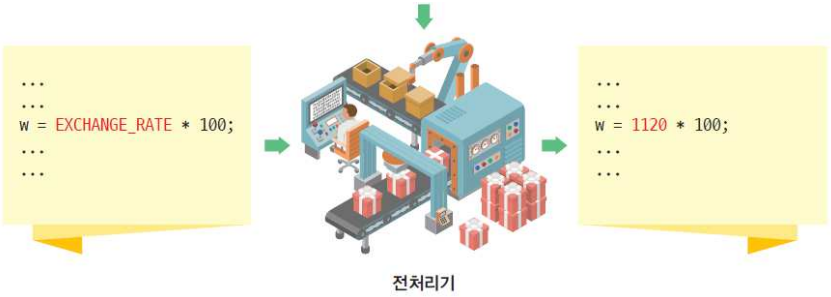
```c
#include <stdio.h>
#define EXCHANGE_RATE 1050

int main(void)
{
  ...
  won1 = EXCHANGE_RATE * dollar1;
  ...
  won2 = EXCHANGE_RATE * dollar2;
  ...
}
```

기호 상수를 사용하는 경우:
기호 상수가 정의된 곳만 수정하면 한다.

기호 상수를 만드는 방법 #1

EXCHANGE_RATE이라는 기호를 1120으로 정의

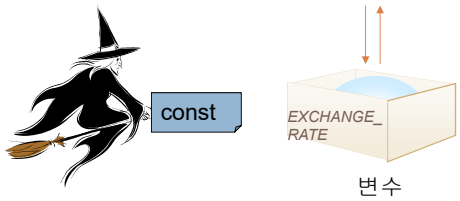
```
#define   EXCHANGE_RATE     1120
```



기호 상수를 만드는 방법 #2

변수가 값을 변경할 수 없게 한다.

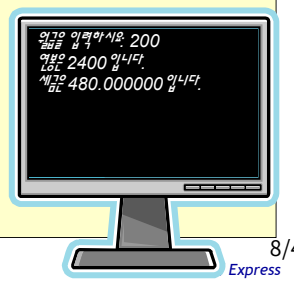
```
const   int EXCHANGE_RATE = 1120;
```



예제: 기호 상수

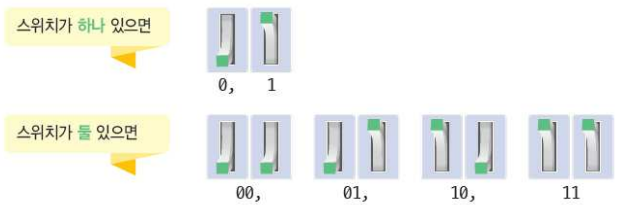
```
#include <stdio.h>
#define TAX_RATE  0.2

int main(void)
{
        const int MONTHS = 12;
        int m_salary, y_salary;          // 변수 선언

        printf( "월급을 입력하시요: ");  // 입력 안내문
        scanf("%d", &m_salary);
        y_salary = MONTHS * m_salary; // 순수입 계산
        printf("연봉은 %d입니다.",  y_salary);
        printf("세금은 %f입니다.",  y_salary*TAX_RATE);

        return 0;
}
```

기호상수



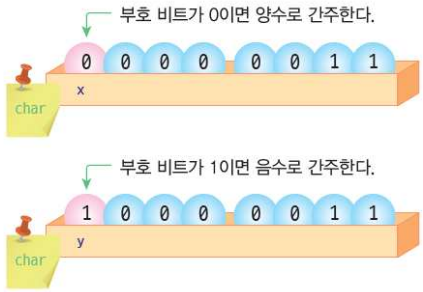
정수 표현 방법

- 컴퓨터에서 정수는 이진수 형태로 표현되고 이진수는 전자 스위치로 표현된다.



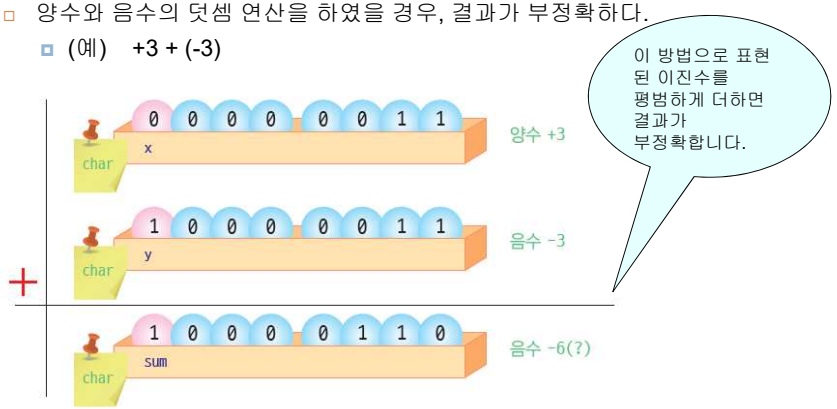
정수 표현 방법

- 양수
  - 십진수를 이진수로 변환하여 저장하면 된다.
- 음수
  - 보통은 첫번째 비트를 부호 비트로 사용한다.
  - 문제점이 발생한다.



음수를 표현하는 첫번째 방법

- 첫번째 방법은 맨 처음 비트를 부호 비트로 간주하는 방법입니다.
- 양수와 음수의 덧셈 연산을 하였을 경우, 결과가 부정확하다.
  - (예) +3 + (-3)



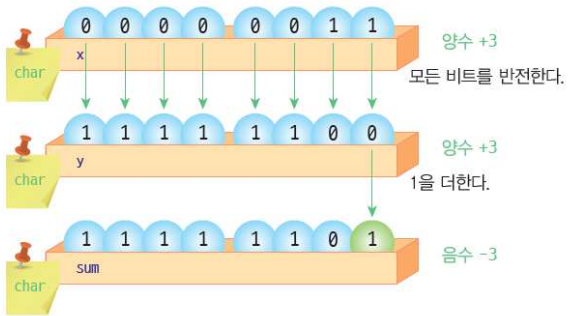
컴퓨터는 덧셈만 할 수 있다

- 컴퓨터는 회로의 크기를 줄이기 위하여 덧셈회로만을 가지고 있다.
- 뺄셈은 다음과 같이 덧셈으로 변환한다.

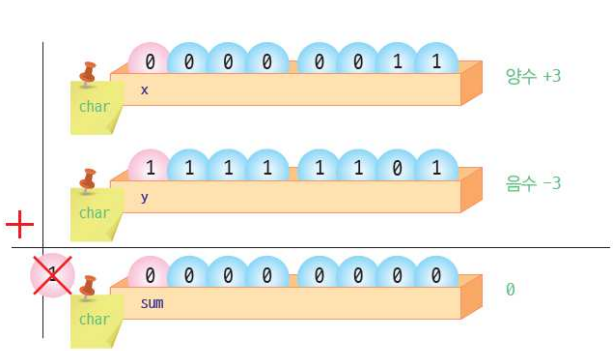
$$3-3 = 3+(-3)$$

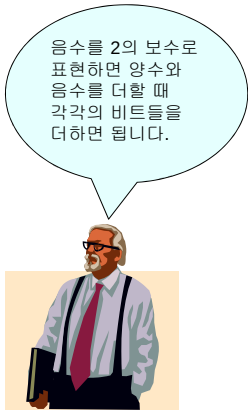
음수를 표현하는 두번째 방법

- 2의 보수로 음수를 표현한다. -> 표준적인 음수 표현 방법
- 2의 보수를 만드는 방법



2의 보수로 양수와 음수를 더하면





예제

```
/* 2의 보수 프로그램*/
#include <stdio.h>

int main(void)
{
        int x = 3;
        int y = -3;

        printf("x = %08X\n", x);          // 8자리의 16진수로 출력한다.
        printf("y = %08X\n", y);          // 8자리의 16진수로 출력한다.
        printf("x+y = %08X\n", x+y);      // 8자리의 16진수로 출력한다.

        return 0;
}
```

음수가 2의 보수로 표현되는지를 알아보자.

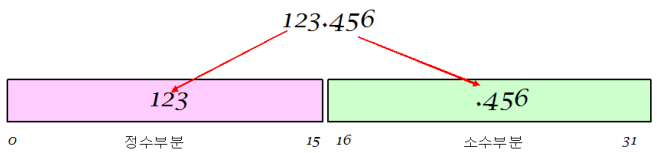
```
x = 00000003
y = FFFFFFFD
x+y = 00000000
```


부동소수점형

- 컴퓨터에서 실수는 부동소수점형으로 표현
  - 소수점이 떠서 움직인다는 의미
  - 과학자들이 많이 사용하는 과학적 표기법과 유사

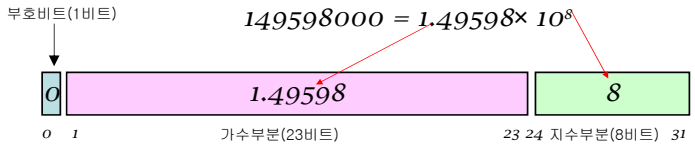
실수를 표현하는 방법

- #1 고정 소수점 방식
  - 정수 부분을 위하여 일정 비트를 할당하고 소수 부분을 위하여 일정 비트를 할당
  - 전체가 32비트이면 정수 부분 16비트, 소수 부분 16비트 할당
  - 과학과 공학에서 필요한 아주 큰 수를 표현할 수 없다



실수를 표현하는 방법

- #2 부동 소수점 방식



- 표현할 수 있는 범위가 대폭 늘어난다.
- 10^{-38} 에서 10^{+38}

부동 소수점 형

자료형	명칭	크기	범위
float	단일 정밀도(single-precision) 부동 소수점	32비트	$\pm 1.17549\times10^{-38}$~$\pm 3.40282\times10^{+38}$
double long double	두배 정밀도(double-precision) 부동 소수점	64비트	$\pm 2.22507\times10^{-308}$~$\pm 1.79769\times10^{+308}$

예제

```c
/* 부동 소수점 자료형의 크기 계산*/
#include <stdio.h>
int main(void)
{
	float x = 1.234567890123456789;
	double y = 1.234567890123456789;

	printf("float의 크기=%d\n", sizeof(float));
	printf("double의 크기=%d\n", sizeof(double));

	printf("x = %30.25f\n",x);
	printf("y = %30.25f\n",y);
	return 0;
}
```

```
float의 크기=4
double의 크기=8
x =    1.2345678806304932000000000
y =    1.2345678901234567000000000
```


부동 소수점 상수

실수	지수 표기법	의미
123.45	1.2345e2	1.2345×10^2
12345.0	1.2345e5	1.2345×10^5
0.000023	2.3e-5	2.3×10^{-5}
2,000,000,000	2.0e9	2.0×10^9

```
1.23456
2.          // 소수점만 붙여도 된다.
.28         // 정수부가 없어도 된다.
2e+10       // +나 -기호를 지수부에 붙일 수 있다.
9.26E3      // 9.26×10^3
0.67e-7     // 0.67×10^-9
```

부동 소수점 오버플로우

```
#include <stdio.h>

int main(void)
{
  float x = 1e39;
  printf("x = %e\n",x);
}
```

숫자가 커서 오버플로우 발생

```
x = 1.#INF00e+000
계속하려면 아무 키나 누르십시오 . . .
```


부동 소수점 언더플로우

```
#include <stdio.h>

int main(void)
{
      float x = 1.23456e-38;
      float y = 1.23456e-40;
      float z = 1.23456e-46;

      printf("x = %e\n",x);
      printf("y = %e\n",y);
      printf("z = %e\n",z);
}
```

숫자가 작아서 언더플로우 발생

```
x = 1.234560e-038
y = 1.234558e-040
z = 0.000000e+000
```


부동소수점형 사용시 주의사항

- 오차가 있을 수 있다!

```
#include <stdio.h>

int main(void)
{
      double x;

      x = (1.0e20 + 5.0)-1.0e20;
      printf("%f \n",x);
      return 0;
}
```

부동소수점 연산에서는 오차가 발생한다. 5.0이 아니라 0으로 계산된다.

```
0.000000
```


중간 점검

- float형과 double형의 크기는?
- 부동 소수점 상수인 1.0× 1025를 지수 형식으로 표기하여 보자.
- 부동 소수점형에서 오차가 발생하는 근본적인 이유는 무엇인가?

문자형

- 문자는 컴퓨터보다는 인간에게 중요
- 문자도 숫자를 이용하여 표현

문자형

- 문자는 컴퓨터보다는 인간에게 중요
- 문자도 숫자를 이용하여 표현
- 공통적인 규격이 필요하다.
- 아스키 코드(**ASCII:** American Standard Code for Information Interchange)

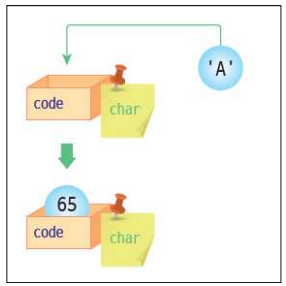
아스키 코드표 (일부)

Dec	Hex	문자	Dec	Hex	문자	Dec	Hex	문자	Dec	Hex	문자
0	0	NULL	20	14	DC4	40	28	(	60	3C	<
1	1	SOH	21	15	NAK	41	29	)	61	3D	=
2	2	STX	22	16	SYN	42	2A	*	62	3E	>
3	3	ETX	23	17	ETB	43	2B	+	63	3F	?
4	4	EOL	24	18	CAN	44	2C	,	64	40	@
5	5	ENQ	25	19	EM	45	2D	-	65	41	A
6	6	ACK	26	1A	SUB	46	2E	.	66	42	B
7	7	BEL	27	1B	ESC	47	2F	/	67	43	C
8	8	BS	28	1C	FS	48	30	0	68	44	D
9	9	HT	29	1D	GS	49	31	1	69	45	E
10	A	LF	30	1E	RS	50	32	2	70	46	F
11	B	VT	31	1F	US	51	33	3	71	47	G
12	C	FF	32	20	space	52	34	4	72	48	H
13	D	CR	33	21	!	53	35	5	73	49	I
14	E	SO	34	22	"	54	36	6	74	4A	J
15	F	SI	35	23	#	55	37	7	75	4B	K
16	10	DLE	36	24	$	56	38	8	76	4C	L
17	11	DC1	37	25	%	57	39	9	77	4D	M
18	12	DC2	38	26	&	58	3A	:	78	4E	N
19	13	DC3	39	27	'	59	3B	;	79	4F	O

문자 변수

- char형을 사용하여 문자를 저장한다.

```
char code;
code = 'A';
```



예제

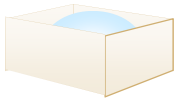
```
/* 문자 변수와 문자 상수*/
#include <stdio.h>

int main(void)
{
        char code1 = 'A';    // 문자 상수로 초기화
        char code2 = 65;     // 아스키 코드로 초기화

        printf("code1 = %c\n", code1);
        printf("code2 = %c\n", code2);
}
```

```
code1 = A
code2 = A
```


Quiz

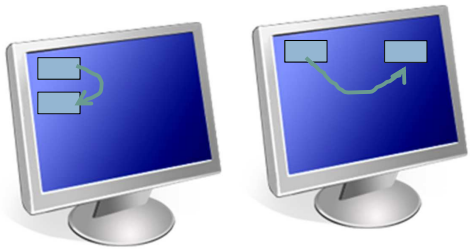
(Q) 1과 '1'의 차이점은?

(A) 1은 정수이고 '1'은 문자 '1'을 나타내는 아스키코드이다.



제어 문자

- 인쇄 목적이 아니라 제어 목적으로 사용되는 문자들
  - (예) 줄바꿈 문자, 탭 문자, 벨소리 문자, 백스페이스 문자



제어 문자를 나타내는 방법

- 아스키 코드를 직접 사용

```
char beep = 7;
printf("%c", beep);
```

- 이스케이프 시퀀스 사용

```
char beep = '\a';
printf("%c", beep);
```

이스케이프 시퀀스

제어 문자	이름	의미
\0	널문자	
\a	경고(bell)	"삐"하는 경고음 발생
\b	백스페이스(backspace)	커서를 현재의 위치에서 한 글자 뒤로 옮긴다.
\t	수평탭(horizontal tab)	커서의 위치를 현재 라인에서 설정된 다음 탭 위치로 옮긴다.
\n	줄바꿈(newline)	커서를 다음 라인의 시작 위치로 옮긴다.
\v	수직탭(vertical tab)	설정되어 있는 다음 수직 탭 위치로 커서를 이동
\f	폼피드(form feed)	주로 프린터에서 강제적으로 다음 페이지로 넘길 때 사용된다.
\r	캐리지 리턴(carriage return)	커서를 현재 라인의 시작 위치로 옮긴다.
\"	큰따옴표	원래의 큰따옴표 자체
\'	작은따옴표	원래의 작은따옴표 자체
\\	역슬래시(back slash)	원래의 역슬래시 자체

예제

```
#include <stdio.h>
int main()
{
	int id, pass;

	printf("아이디와 패스워드를 4개의 숫자로 입력하세요:\n");

	printf("id: ____\b\b\b\b");
	scanf("%d", &id);

	printf("pass: ____\b\b\b\b");
	scanf("%d", &pass);
	printf("\a입력된 아이디는 \"%d\"이고 패스워드는 \"%d\"입니다.", id, pass);

	return 0;
}
```

```
아이디와 패스워드를 4 개의 숫자로 입력하세요:
id: 1234
pass: 5678
입력된 아이디는 "1234"이고 패스워드는 "5678"입니다.
```


정수형으로서의 char형

- 8비트의 정수를 저장하는데 char 형을 사용할 수 있다..

```
#include <stdio.h>

int main()
{
	char code = 'A';
	printf("%d %d %d \n", code, code+1, code+2); // 65 66 67이 출력된다.
	printf("%c %c %c \n", code, code+1, code+2); // A B C가 출력된다.
	return 0;
}
```

```
65 66 67
A B C
```

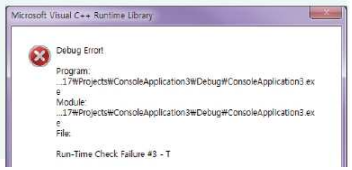

중간 점검

- 컴퓨터에서는 문자를 어떻게 나타내는가?
- C에서 문자를 가장 잘 표현할 수 있는 자료형은 무엇인가?
- 경고음이 발생하는 문장을 작성하여 보자.

무엇이 문제일까?

sum_error.c

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int x, y, z, sum;
5     printf("3개의 정수를 입력하세요 (x, y, z): ");
6     scanf("%d %d %d", &x, &y, &z);
7     sum += x;
8     sum += y;
9     sum += z;
10    printf("3개 정수의 합은 %d\n", sum);
11    return 0;
12 }
```



무엇이 문제일까?

sum_error.c

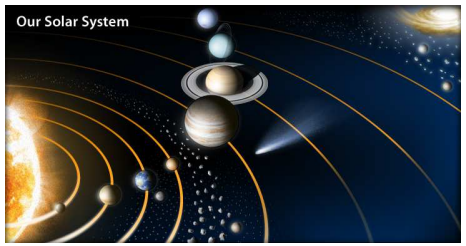
```
1  #include <stdio.h>
2  int main(void)
3  {
4     int x, y, z, sum;
5
6     sum = 0;
7     printf("3개의 정수를 입력하세요 (x, y, z): ");
8     scanf("%d %d %d", &x, &y, &z);
9     sum += x;
10    sum += y;
11    sum += z;
12    printf("3개 정수의 합은 %d\n", sum);
13    return 0;
14 }
```

실행결과

```
3개의 정수를 입력하세요 (x, y, z): 10 20 30
3개의 정수의 합은 60
```


mini project: 태양빛 도달 시간

- 태양에서 오는 빛이 몇 분 만에 지구에 도착하는 지를 컴퓨터로 계산해보고자 한다.
- 빛의 속도는 1초에 30만 km를 이동한다.
- 태양과 지구 사이의 거리는 약 1억 4960만 km이다.



실행 결과

힌트

- 문제를 해결하기 위해서는 먼저 필요한 변수를 생성하여야 한다. 여기서는 빛의 속도, 태양과 지구 사이의 거리, 도달 시간을 나타내는 변수가 필요하다.
- 변수의 자료형은 모두 실수형이어야 한다. 왜냐하면 매우 큰 수들이기 때문이다.
- 빛이 도달하는 시간은 (도달 시간 = 거리/ (빛의 속도))으로 계산할 수 있다.
- 실수형을 printf()로 출력할 때는 %f나 %lf를 사용한다.

소스

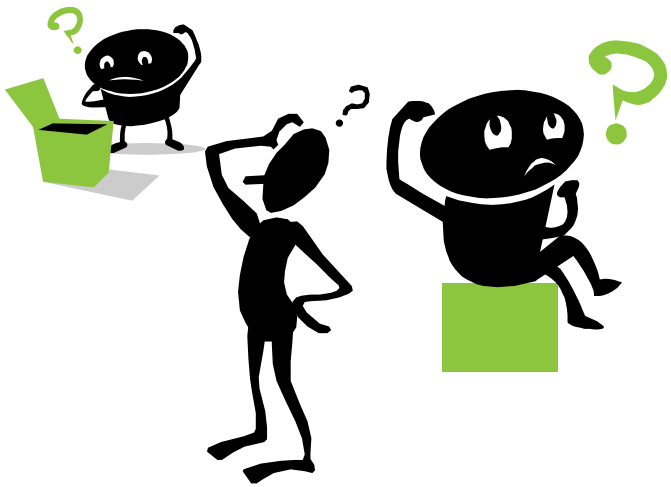
```
#include <stdio.h>
int main(void)
{
        double light_speed = 300000;        // 빛의 속도 저장하는 변수
        double distance = 149600000;        // 태양과 지구 사이 거리 저장하는 변수
                                            // 149600000km로 초기화한다.
        double time;                        // 시간을 나타내는 변수

        time = distance / light_speed;      // 거리를 빛의 속도로 나눈다.
        time = time / 60.0;                 // 초를 분으로 변환한다.

        printf("빛의 속도는 %fkm/s \n", light_speed);
        printf("태양과 지구와의 거리 %fkm \n", distance);
        printf("도달 시간은 %f초\n", time);        // 시간을 출력한다.

        return 0;
}
```

```
빛의 속도는 300000.000000km/s
태양과 지구와의 거리 149600000.000000km
도달 시간은 498.666667초
```


도전문제

- 위의 프로그램의 출력은 498.666667초로 나온다. 이것을 분과 초로 나누어서 8분 20초와 같은 식으로 출력하도록 변경하라. 나머지를 계산하는 연산자는 %이다. 추가적인 정수 변수를 사용하여도 좋다.

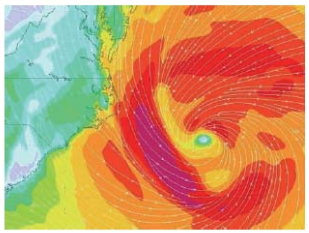
Q & A



이번 장에서 학습할 내용

* 수식과 연산자란?
* 대입 연산
* 산술 연산
* 논리 연산
* 관계 연산
* 우선 순위와 결합 법칙

컴퓨터는 근본적으로 계산하는 기계

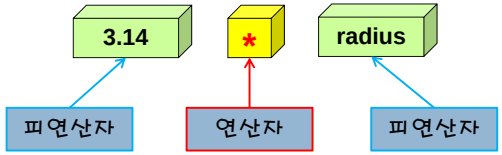


수식의 예


```
int x, y;

x = 3;
y = x*x - 5*x + 6;
printf("%d\n", y);
```

수식

- 수식(expression)
  - 상수, 변수, 연산자의 조합
  - 연산자와 피연산자로 나누어진다.



기능에 따른 연산자의 분류

연산자의 분류	연산자	의미
대입	=	오른쪽을 왼쪽에 대입
산술	+ - * / %	사칙연산과 나머지 연산
부호	+ -	
증감	++ --	증가, 감소 연산
관계	> < == != >= <=	오른쪽과 왼쪽을 비교
논리	&& || !	논리적인 AND, OR, NOT
조건	?	조건에 따라 선택
콤마	,	피연산자들을 순차적으로 실행
비트 단위 연산자	& | ^ ~ << >>	비트별 AND, OR, XOR, 반전, 이동
sizeof 연산자	sizeof	자료형이나 변수의 크기를 바이트 단위로 반환
형변환	(type)	변수나 상수의 자료형을 변환
포인터 연산자	* & []	주소계산, 포인터가 가리키는 곳의 내용 추출
구조체 연산자	. ->	구조체의 멤버 참조

피연산자수에 따른 연산자 분류

- 단항 연산자: 피연산자의 수가 1개

```
++x;
--y;
```

- 이항 연산자: 피연산자의 수가 2개

```
x + y
x - y
```

- 삼항 연산자: 연산자의 수가 3개

```
x ? y : z
```


중간 점검

1. 수식(expression)이란 어떻게 정의되는가?
2. 상수 10도 수식이라고 할 수 있는가?
3. 아래의 수식에서 피연산자와 연산자를 구분하여 보라.

   y = 10 + 20;
4. 연산자를 단항 연산자, 이항 연산자, 삼항 연산자로 나누는 기준은 무엇인가?

산술 연산자

- 산술 연산: 컴퓨터의 가장 기본적인 연산
- 덧셈, 뺄셈, 곱셈, 나눗셈 등의 사칙 연산을 수행하는 연산자

연산자	기호	사용예	결과값
덧셈	+	7 + 4	11
뺄셈	–	7 – 4	3
곱셈	*	7 * 4	28
나눗셈	/	7 / 4	1
나머지	%	7 % 4	3

산술 연산자의 예

$y = mx + b$ → $y = m*x + b$

$y = ax^2 + bx + c$ → $y = a*x*x + b*x + c$

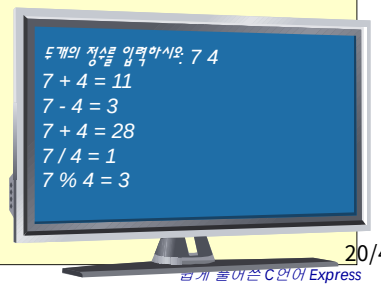
$m = \frac{x + y + x}{3}$ → $m = (x + y + z) / 3$

(참고) 거듭 제곱 연산자는?

C에는 거듭 제곱을 나타내는 연산자는 없다.
x * x와 같이 단순히 변수를 두 번 곱한다.

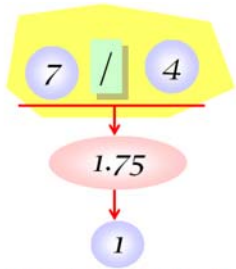
정수 사칙 연산

```
#include <stdio.h>

int main(void)
{
      int x, y, result;
      printf("두개의 정수를 입력하시오: ");
      scanf("%d %d", &x, &y);

      result = x + y;
      printf("%d + %d = %d", x, y, result);

      result = x - y;                  // 뺄셈
      printf("%d - %d = %d", x, y, result);
      result = x * y;                  // 곱셈
      printf("%d + %d = %d", x, y, result);
      result = x / y;                  // 나눗셈
      printf("%d / %d = %d", x, y, result);
      result = x % y;                  // 나머지
      printf("%d %% %d = %d", x, y, result);
      return 0;
}
```

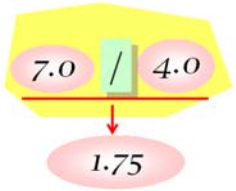


나눗셈 연산자

- 정수형끼리의 나눗셈에서는 결과가 정수형으로 생성하고 부동소수점형끼리는 부동소수점 값을 생성된다.
- 정수형끼리의 나눗셈에서는 소수점 이하는 버려진다.



정수와 정수 끼리의 나눗셈.



실수와 실수 끼리의 나눗셈.

실수 사칙 연산

```c
#include <stdio.h>

int main()
{
    double x, y, result;

    printf("두개의 실수를 입력하시오: ");
    scanf("%lf %lf", &x, &y);

    result = x + y;              // 덧셈 연산을 하여서 결과를 result에 대입
    printf("%f / %f = %f", x, y, result);

        ...
    result = x / y;
    printf("%f / %f = %f", x, y, result);

    return 0;
}
```

```
두개의 실수를 입력하시오: 7 4
7.000000 + 4.000000 = 11.000000
7.000000 - 4.000000 = 3.000000
7.000000 + 4.000000 = 28.000000
7.000000 / 4.000000 = 1.750000
```


나머지 연산자

- 나머지 연산자(modulus operator)는 첫 번째 피연산자를 두 번째 피연산자로 나누었을 경우의 나머지를 계산
  - 10 % 2는 0이다.
  - 5 % 7는 5이다.
  - 30 % 9는 3이다.
- 나머지 연산자를 이용한 짝수와 홀수를 구분
  - x % 2가 0이면 짝수
- 나머지 연산자를 이용한 5의 배수 판단
  - x % 5가 0이면 5의 배수

나머지 연산자

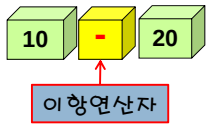
```c
// 나머지 연산자 프로그램
#include <stdio.h>
#define SEC_PER_MINUTE 60  // 1분은 60초

int main(void)
{
    int input, minute, second;

    printf( " 초를 입력하시요: ");
    scanf("%d", &input);        // 초단위의 시간을 읽는다.

    minute = input / SEC_PER_MINUTE; // 몇 분
    second = input % SEC_PER_MINUTE; // 몇 초

    printf("%d초는 %d분 %d초입니다. \n",
            input, minute, second);
    return 0;
}
```

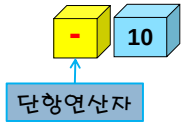

```
초를 입력하시요: 1000
1000초는 16분 40초 입니다.
```

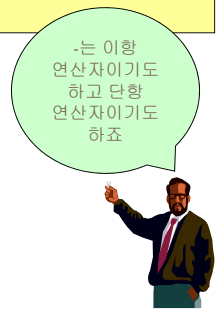

부호 연산자

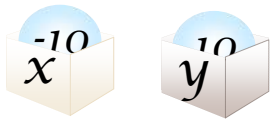
- 변수나 상수의 부호를 변경

```c
x = -10;
y = -x;   // 변수 y의 값은 10이 된다.
```







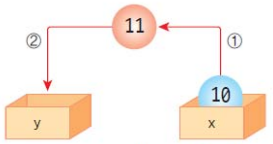


증감 연산자

- 증감 연산자: ++, --
- 변수의 값을 하나 증가시키거나 감소시키는 연산자
- ++x, --x;

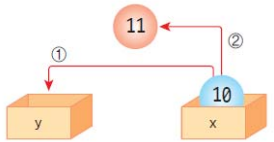
++x와 x++의 차이

y=++x;



증가된 x의 값이 y에 대입된다.

y=x++;



먼저 대입하고 나중에 증가한다.

증감 연산자 정리

증감 연산자	의미
++x	수식의 값은 증가된 x값이다.
x++	수식의 값은 증가되지 않은 원래의 x값이다.
--x	수식의 값은 감소된 x값이다.
x--	수식의 값은 감소되지 않은 원래의 x값이다.

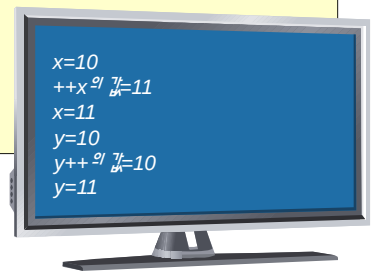
Quiz

- nextx와 nexty의 값은?

```
x = 1;
y = 1;

nextx = ++x;
nexty = y++;
```

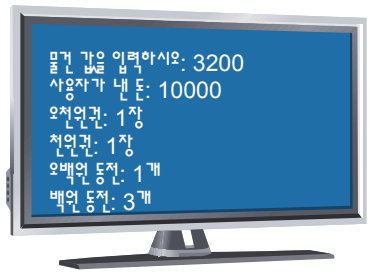
예제: 증감 연산자

```
#include <stdio.h>
int main(void)
{
        int x=10, y=10;

        printf("x=%d\n", x);
        printf("++x의 값=%d\n", ++x);
        printf("x=%d\n\n", x);

        printf("y=%d\n", y);
        printf("y++의 값=%d\n", y++);
        printf("y=%d\n", y);

        return 0;
}
```



Lab: 거스름돈 계산하기

- 편의점에서 물건을 구입하고 만 원을 냈을 때, 거스름돈의 액수와 점원이 지급해야 할 거스름돈을 화폐와 동전수를 계산하는 프로그램을 작성해보자.




```
#include <stdio.h>
int main(void)
{
        int user, change = 0;
        int price, c5000, c1000, c500, c100;

        printf("물건 값을 입력하시오: ");
        scanf("%d", &price); // 물건 값을 입력받는다.
        printf("사용자가 낸 돈: ");
        scanf("%d", &user);
        change = user - price;          // 거스름돈을 change에 저장
```



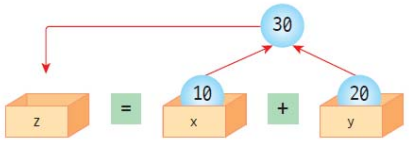
```
        c5000 = change / 5000; // 몫 연산자를 사용하여 5000원권의 개수를 계산한다.
        change = change % 5000; // 나머지 연산자를 사용하여 남은 잔돈을 계산한다.

        c1000 = change / 1000; // 남은 잔돈에서 1000원권의 개수를 계산한다.
        change = change % 1000; //나머지 연산자를 사용하여 남은 잔돈을 계산한다.

        c500 = change / 500; // 남은 잔돈에서 500원 동전의 개수를 계산한다.
        change = change % 500; //나머지 연산자를 사용하여 남은 잔돈을 계산한다.

        c100 = change / 100; // 남은 잔돈에서 100원 동전의 개수를 계산한다.
        change = change % 100; //나머지 연산자를 사용하여 남은 잔돈을 계산한다.

        printf("오천원권: %d장\n", c5000);
        printf("천원권: %d장\n", c1000);
        printf("오백원 동전: %d개\n", c500);
        printf("백원 동전: %d개\n", c100);
        return 0;
}
```

대입(배정, 할당) 연산자

Syntax: 대입 연산자

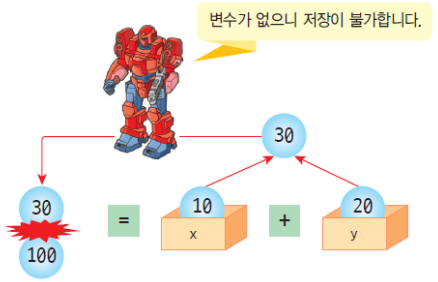
예 z = x + y

변수 수식



대입 연산자 주의점

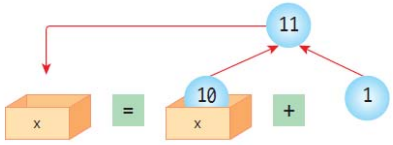
- 100 = x + y; // 컴파일 오류!



대입 연산자 주의점

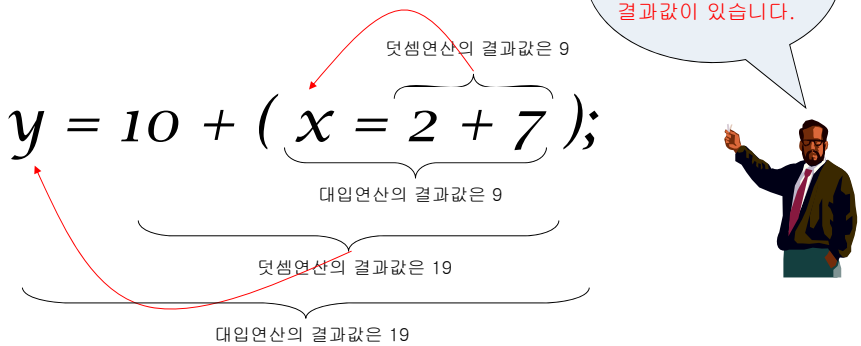
수학적으로는 올바르지 않지만 c에서는 올바른 문장임

x = x + 1;



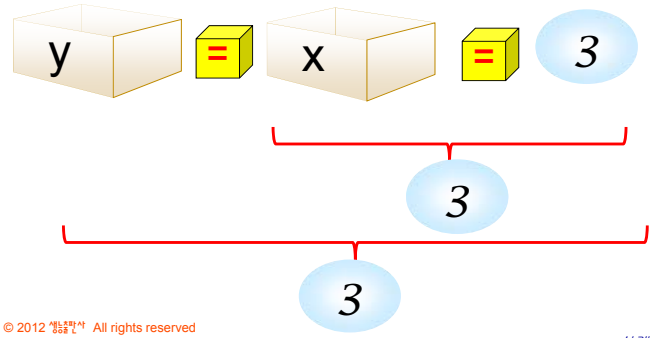
대입 연산의 결과값

모든 연산에는 결과값이 있고 대입 연산도 결과값이 있습니다.



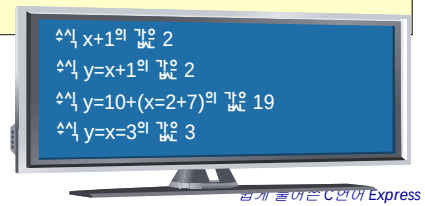
예제

```
y = x = 3;
```



예제

```
/* 대입 연산자 프로그램 */
#include <stdio.h>

int main(void)
{
        int x, y;

        x = 1;
        printf("수식 x+1의 값은 %d\n", x+1);
        printf("수식 y=x+1의 값은 %d\n", y=x+1);
        printf("수식 y=10+(x=2+7)의 값은 %d\n", y=10+(x=2+7));
        printf("수식 y=x=3의 값은 %d\n", y=x=3);

        return 0;
}
```



복합 대입 연산자

- 복합 대입 연산자란 +=처럼 대입연산자 =와 산술연산자를 합쳐 놓은 연산자
- 소스를 간결하게 만들 수 있음

x = x + y와 의미가 같음!

x += y

복합 대입 연산자

복합 대입 연산자	의미
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

Quiz

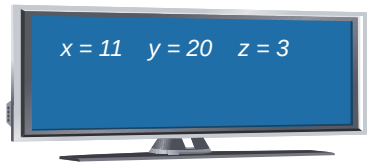
- 다음 수식을 풀어서 다시 작성하면?

```
x *= y + 1
x %= x + y
```

```
x = x * (y + 1)
x = x % (x + y)
```

복합 대입 연산자

```
// 복합 대입 연산자 프로그램
#include <stdio.h>

int main(void)
{
        int x = 10, y = 10, z = 33;

        x += 1;
        y *= 2;
        z %= 10 + 20;

        printf("x = %d    y = %d    z = %d \n", x, y, z);
        return 0;
}
```



관계 연산자

- 두개의 피연산자를 비교하는 연산자
- 결과값은 참(1) 아니면 거짓(0)

x == y

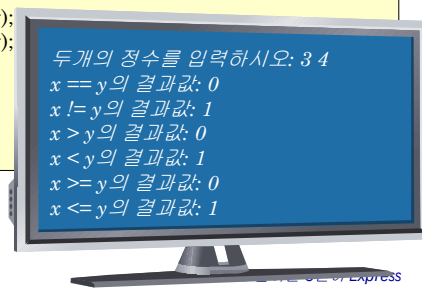
x 와 y의 값이 같은지 비교한다.

관계 연산자

연산자	의미
x == y	x와 y가 같은가?
x != y	x와 y가 다른가?
x > y	x가 y보다 큰가?
x < y	x가 y보다 작은가?
x >= y	x가 y보다 크거나 같은가?
x <= y	x가 y보다 작거나 같은가?

예제

```
#include <stdio.h>
int main(void)
{
      int x, y;

      printf("두개의 정수를 입력하시오: ");
      scanf("%d%d", &x, &y);

      printf("x == y의 결과값: %d", x == y);
      printf("x != y의 결과값: %d", x != y);
      printf("x > y의 결과값: %d", x > y);
      printf("x < y의 결과값: %d", x < y);
      printf("x >= y의 결과값: %d", x >= y);
      printf("x <= y의 결과값: %d", x <= y);

      return 0;
}
```

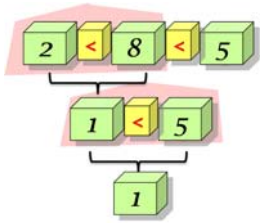


주의할 점!

- (x = y)
  - y의 값을 x에 대입한다. 이 수식의 값은 x의 값이다.
- (x == y)
  - x와 y가 같으면 1, 다르면 0이 수식의 값이 된다.
  - (x == y)를 (x = y)로 잘못 쓰지 않도록 주의!

관계 연산자 사용시 주의점

- 수학에서처럼 2 < x < 5와 같이 작성하면 잘못된 결과가 나온다.



- 올바른 방법: (2 < x) && (x < 5)

실수를 비교하는 경우

- (1e32 + 0.01) > 1e32
  - -> 양쪽의 값이 같은 것으로 간주되어서 거짓

중간 점검

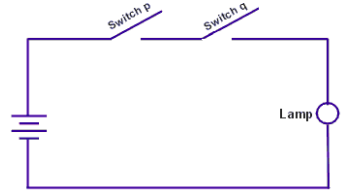
1. 관계 수식의 결과로 생성될 수 있는 값은 무엇인가?
2. (3 >= 2) + 5의 값은?

논리 연산자

- 여러 개의 조건을 조합하여 참과 거짓을 따지는 연산자
- 결과값은 참(1) 아니면 거짓(0)

x와 y가 모두 참인 경우에만 참이 된다.

x && y



논리 연산자

연산자	의미
x && y	AND 연산, x와 y가 모두 참이면 참, 그렇지 않으면 거짓
x || y	OR 연산, x나 y중에서 하나만 참이면 참, 모두 거짓이면 거짓
!x	NOT 연산, x가 참이면 거짓, x가 거짓이면 참

AND 연산자

27 800

(age <= 30) && (toeic >= 700)

참(1) 참(1)

참(1)

OR 연산자

27 699

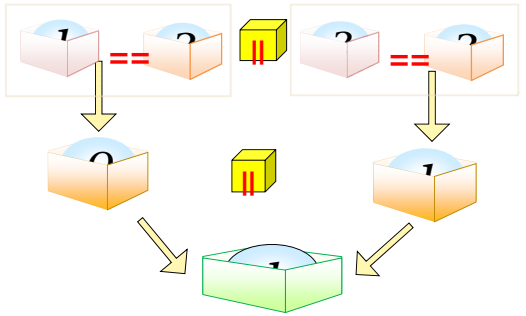
(age <= 30) || (toeic >= 700)

참(1) 거짓(0)

참(1)

논리 연산자의 계산 과정

- 논리 연산의 결과값은 항상 1 또는 0이다.
- (예) (1 == 2) || (2 == 2)



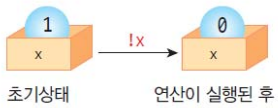
참과 거짓의 표현 방법

- 관계 수식이나 논리 수식이 만약 참이면 1이 생성되고 거짓이면 0이 생성된다.
- 피연산자의 참, 거짓을 가릴 때에는 0이 아니면 참이고 0이면 거짓으로 판단한다.
- 음수는 거짓으로 판단한다.
- (예) NOT 연산자를 적용하는 경우

```
!0              // 식의 값은 1
!3              // 식의 값은 0
!-3             // 식의 값은 0
```


NOT 연산자

- 피연산자의 값이 참이면 연산의 결과값을 거짓으로 만들고, 피연산자의 값이 거짓이면 연산의 결과값을 참으로 만든다.



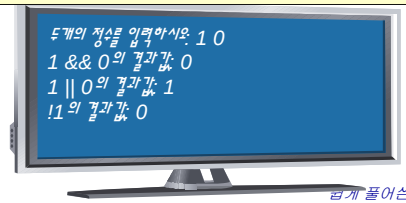
- result = !1; // result에는 0가 대입된다.
- result = !(2==3); // result에는 1이 대입된다.

논리 연산자의 예

- "x는 1, 2, 3중의 하나인가"
  - (x == 1) || (x == 2) || (x == 3)
- "x가 60이상 100미만이다."
  - (x >= 60) && (x < 100)
- "x가 0도 아니고 1도 아니다."
  - (x != 0) && (x != 1) // x≠0 이고 x≠1이다.

예제

```
#include <stdio.h>

int main(void)
{
    int x, y;

    printf("두개의 정수를 입력하시오: ");
    scanf("%d%d", &x, &y);

    printf("%d && %d의 결과값: %d", x, y, x && y);
    printf("%d || %d의 결과값: %d", x, y, x || y);
    printf("!%d의 결과값: %d", x, !x);

    return 0;
}
```



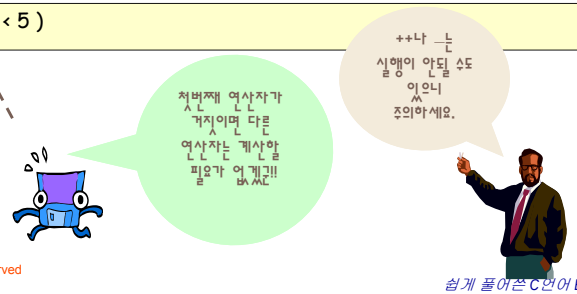
단축 계산

- && 연산자의 경우, 첫번째 피연산자가 거짓이면 다른 피연산자들을 계산하지 않는다.

```
( 2 > 3 ) && ( ++x < 5 )
```

- || 연산자의 경우, 첫번째 피연산자가 참이면 다른 피연산자들을 계산하지 않는다.

```
( 3 > 2 ) || ( --x < 5 )
```

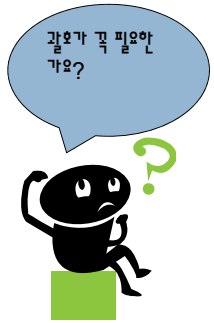


lab: 윤년

- 윤년의 조건
  - 연도가 4로 나누어 떨어진다.
  - 100으로 나누어 떨어지는 연도는 제외한다.
  - 400으로 나누어 떨어지는 연도는 윤년이다.

실습: 윤년

- 윤년의 조건을 수식으로 표현
  - ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)



Lab: 윤년

```
#include <stdio.h>
int main(void)
{
        int year, result;

        printf("연도를 입력하시오: ");
        scanf("%d", &year);

        result = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
        printf("result=%d \n", result);

        return 0;
}
```

중간 점검

1. 다음의 조건에 해당하는 논리 연산식을 만들어 보시오. 변수는 적절하게 선언되어 있다고 가정한다.

"무주택 기간 3년 이상, 가구주의 연령이 40세 이상, 가족의 수가 3명 이상"

2. 상수 10은 참인가 거짓인가?
3. 수식 !3의 값은?
4. 단축 계산의 예를 들어보라.

조건 연산자

x > y 가 참이면 x가 수식의 값이 된다.

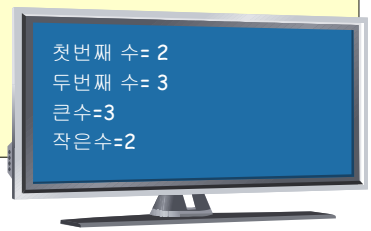
max_value = (x > y) ? x : y;

x > y 가 거짓이면 y가 수식의 값이 된다.

```
absolute_value = (x > 0) ? x: -x;          // 절대값 계산
max_value = (x > y) ? x: y;                // 최대값 계산
min_value = (x < y) ? x: y;                // 최소값 계산
(age > 20) ? printf("성인\n"): printf("청소년\n");
```

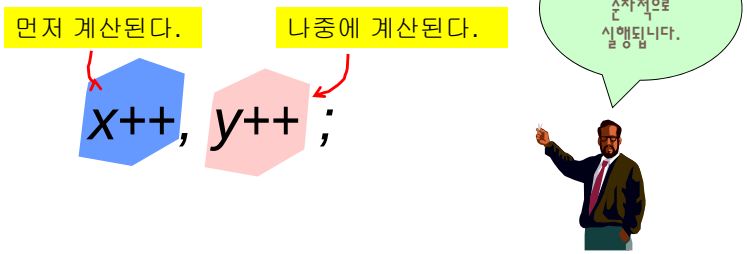
예제

```
#include <stdio.h>
int main(void)
{
        int x,y;

        printf("첫번째 수=");
        scanf("%d", &x);
        printf("두번째 수=");
        scanf("%d", &y);


        printf("큰수=%d \n", (x > y) ? x : y);
        printf("작은수=%d \n", (x < y) ? x : y);

        return 0;
}
```



콤마 연산자

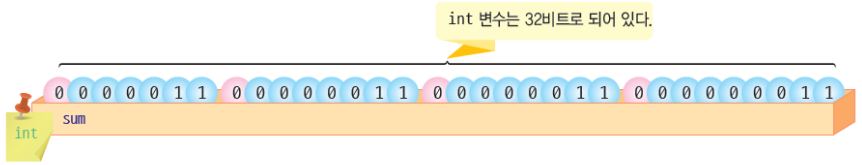
- 콤마로 연결된 수식은 순차적으로 계산된다.



비트 연산자

연산자	연산자의 의미	예
&	비트 AND	두개의 피연산자의 해당 비트가 모두 1이면 1, 아니면 0
|	비트 OR	두개의 피연산자의 해당 비트중 하나만 1이면 1, 아니면 0
^	비트 XOR	두개의 피연산자의 해당 비트의 값이 같으면 0, 아니면 1
<<	왼쪽으로 이동	지정된 개수만큼 모든 비트를 왼쪽으로 이동한다.
>>	오른쪽으로 이동	지정된 개수만큼 모든 비트를 오른쪽으로 이동한다.
~	비트 NOT	0은 1로 만들고 1은 0로 만든다.

모든 데이터는 비트로 이루어진다.



비트 AND 연산자

0 AND 0 = 0
1 AND 0 = 0
0 AND 1 = 0
1 AND 1 = 1

```
변수1 00000000 00000000 00000000 00001001 (9)
변수2 00000000 00000000 00000000 00001010 (10)
-----------------------------------------------------
(변수1 AND 변수2)  00000000 00000000 00000000 00001000 (8)
```


비트 OR 연산자

0 OR 0 = 0
1 OR 0 = 1
0 OR 1 = 1
1 OR 1 = 1

```
변수1 00000000 00000000 00000000 00001001 (9)
변수2 00000000 00000000 00000000 00001010 (10)
-----------------------------------------------------
(변수1 OR 변수2)  00000000 00000000 00000000 00001011 (11)
```


비트 XOR 연산자

0 XOR 0 = 0
1 XOR 0 = 1
0 XOR 1 = 1
1 XOR 1 = 0

```
변수1 00000000 00000000 00000000 00001001 (9)
변수2 00000000 00000000 00000000 00001010 (10)
-----------------------------------------------------
(변수1 XOR 변수2) 00000000 00000000 00000000 00000011 (3)
```


비트 NOT 연산자

NOT 0 = 1
NOT 1 = 0

부호비트가 반전되었기 때문에 음수가 된다.

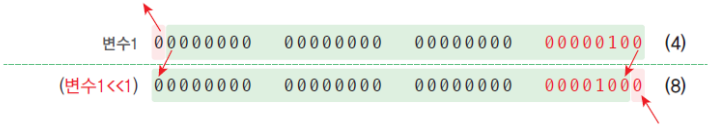
```
변수1 00000000 00000000 00000000 00001001 (9)
-----------------------------------------------------
(NOT 변수1) 11111111 11111111 11111111 11110110 (-10)
```

비트 이동 연산자

연산자	기호	설명
왼쪽 비트 이동	<<	x << y x의 비트들을 y 칸만큼 왼쪽으로 이동
오른쪽 비트 이동	>>	x >> y x의 비트들을 y 칸만큼 오른쪽으로 이동

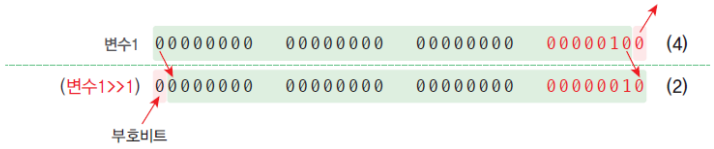
<< 연산자

- 비트를 왼쪽으로 이동
- 값은 2배가 된다.



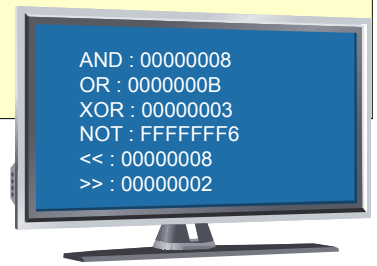
>> 연산자

- 비트를 오른쪽으로 이동
- 값은 1/2배가 된다.



예제: 비트 연산자

```
#include <stdio.h>

int main(void)
{
        printf("AND : %08X\n", 0x9 & 0xA);
        printf("OR : %08X\n", 0x9 | 0xA);
        printf("XOR : %08X\n", 0x9 ^ 0xA);
        printf("NOT : %08X\n", ~0x9);
        printf("<< : %08X\n", 0x4 << 1);
        printf(">> : %08X\n", 0x4 >> 1);
        return 0;
}
```



Lab: 10진수를 2진수로 출력하기

- 비트 연산자를 이용하여 128보다 작은 10진수를 2진수 형식으로 화면에 출력해보자.

Lab: 10진수를 2진수로 출력하기

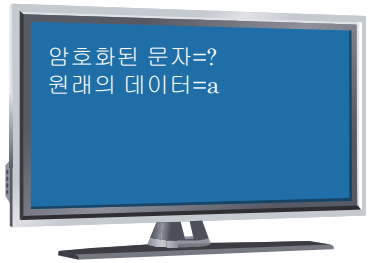
```
#include<stdio.h>

int main(void)
{
        unsigned int num;
        printf("십진수: ");
        scanf("%u", &num);

        unsigned int mask = 1 << 7;   // mask = 10000000
        printf("이진수: ");

        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;             // 오른쪽으로 1비트 이동한다.
        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;             // 오른쪽으로 1비트 이동한다.
        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;             // 오른쪽으로 1비트 이동한다.
```


Lab: 10진수를 2진수로 출력하기

```
        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;
        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;
        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;
        ((num & mask) == 0) ? printf("0") : printf("1");
        mask = mask >> 1;
        ((num & mask) == 0) ? printf("0") : printf("1");
        printf("\n");

        return 0;
}
```


Lab: XOR를 이용한 암호화

- 하나의 문자를 암호화하기 위해서는 x=x^key;하면 된다. 복호화도 x=x^key;하면 된다.

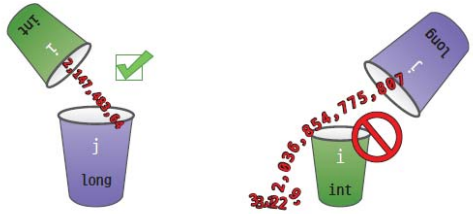


Lab: XOR를 이용한 암호화

```
#include <stdio.h>
int main(void)
{
        char data = 'a';
        char key = 0xff;

        char encrpted_data;
        encrpted_data = data ^ key;

        printf("암호화된 문자=%c \n", encrpted_data);

        char orig_data;
        orig_data = encrpted_data ^ key;
        printf("원래의 데이터=%c\n", orig_data);

        return 0;
}
```


형변환

- 형변환(type conversion)이란 실행 중에 데이터의 타입을 변경하는 것이다



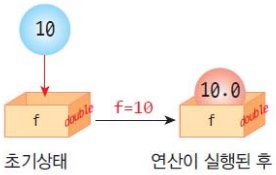
형변환

- 연산시에 데이터의 유형이 변환되는 것

대입 연산시의 자동적인 형변환

- 올림 변환

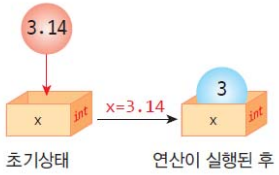
```
double f;
f = 10 ;     // f에는 10.0이 저장된다.
```



대입 연산시의 자동적인 형변환

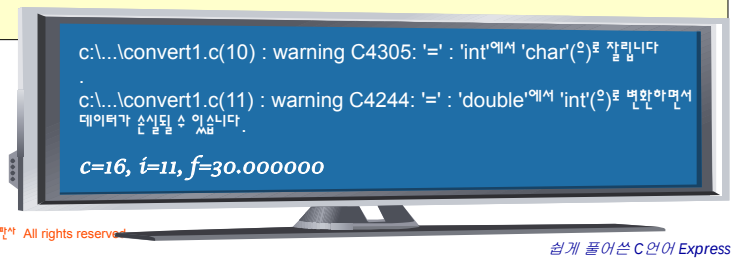
- 내림변환

```
int i;
i = 3.141592;          // i에는 3이 저장된다.
```



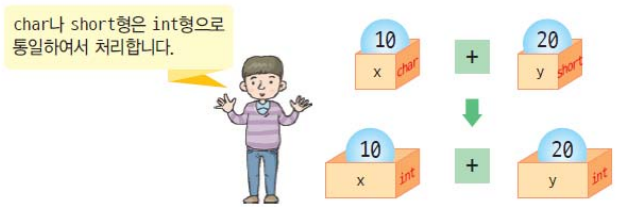
올림 변환과 내림 변환

```
#include <stdio.h>
int main(void)
{
        char c;
        int i;
        float f;

        c = 10000;          // 내림 변환
        i = 1.23456 + 10;   // 내림 변환
        f = 10 + 20;        // 올림 변환
        printf("c = %d, i = %d, f = %f \n", c, i, f);
        return 0;
}
```



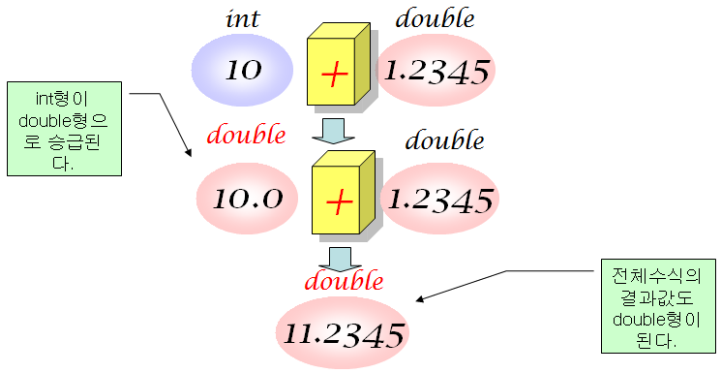
정수 연산시의 자동적인 형변환

- 정수 연산시 char형이나 short형의 경우, 자동적으로 int형으로 변환하여 계산한다.



수식에서의 자동적인 형변환

- 서로 다른 자료형이 혼합하여 사용되는 경우, 더 큰 자료형으로 통일된다.

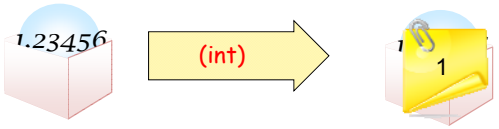


명시적인 형변환

Syntax: 형변환

자료형 수식

```
예  (int)1.23456     // int형으로 변환
    (double) x       // double형으로 변환
    (long) (x+y)     // long형으로 변환
```



예제

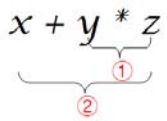
```
#include <stdio.h>

int main(void)
{
        int i;
        double f;

        f = 5 / 4;

        printf("%f\n", f);

        f = (double)5/ 4;
        printf("%f\n", f);

        f = 5.0 / 4;
        printf("%f\n", f);
```

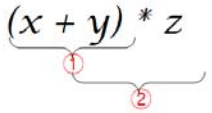

예제

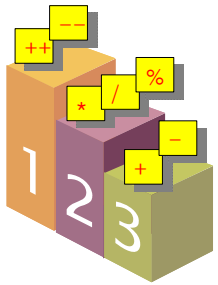
```
        f = (double)5/ (double)4;
        printf("%f\n", f);

        i = 1.3 + 1.8;
        printf("%d\n", i);

        i = (int)1.3+ (int)1.8;

        printf("%d\n", i);
        return 0;
}
```

우선 순위

- 어떤 연산자를 먼저 계산할 것인지에 대한 규칙







우선 순위

우선순위	연산자	설명	결합성
1	++ --	후위 증감 연산자	→ (좌에서 우)
	()	함수 호출	
	[]	배열 인덱스 연산자	
	.	구조체 멤버 접근	
	->	구조체 포인터 접근	
	(type){list}	복합 리터럴(C99 규격)	
2	++ --	전위 증감 연산자	← (우에서 좌)
	+ -	양수, 음수 부호	
	! ~	논리적인 부정, 비트 NOT	
	(type)	형변환	
	*	간접 참조 연산자	
	&	주소 추출 연산자	
	sizeof	크기 계산 연산자	
	_Alignof	정렬 요구 연산자 (C11 규격)	

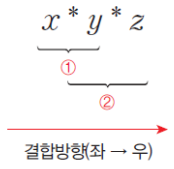
3	* / %	곱셈, 나눗셈, 나머지	→ (좌에서 우)
4	+ -	덧셈, 뺄셈	
5	<< >>	비트 이동 연산자	
6	< <=	관계 연산자	
	> >=	관계 연산자	
7	== !=	관계 연산자	
8	&	비트 AND	
9	^	비트 XOR	
10	|	비트 OR	
11	&&	논리 AND 연산자	
12	||	논리 OR 연산자	
13	?:	삼항 조건 연산자	← (우에서 좌)
14	=	대입 연산자	
	+= -=	복합 대입 연산자	
	*= /= %=	복합 대입 연산자	
	<<= >>=	복합 대입 연산자	
	&= ^= |=	복합 대입 연산자	
15	,	콤마 연산자	→ (좌에서 우)

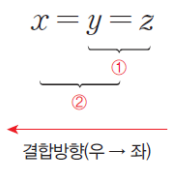
우선 순위의 일반적인 지침

- 콤마 < 대입 < 논리 < 관계 < 산술 < 단항
- 괄호 연산자는 가장 우선순위가 높다.
- 모든 단항 연산자들은 이항 연산자들보다 우선순위가 높다.
- 콤마 연산자를 제외하고는 대입 연산자가 가장 우선순위가 낮다.
- 연산자들의 우선 순위가 생각나지 않으면 괄호를 이용
  - (x <= 10) && (y >= 20)
- 관계 연산자나 논리 연산자는 산술 연산자보다 우선순위가 낮다.
  - x + 2 == y + 3

결합 규칙

- 만약 같은 우선순위를 가지는 연산자들이 여러 개가 있으면 어떤 것을 먼저 수행하여야 하는가의 규칙





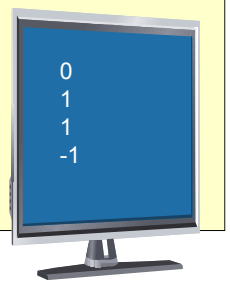
결합규칙의 예

$$y = a \;\%\; b \;/\; c + d * (e - f);$$

② ③ ⑤ ④ ① ⑥

예제

```
#include <stdio.h>
int main(void)
{
    int x=0, y=0;
    int result;

    result = 2 > 3 || 6 > 7;
    printf("%d", result);

    result = 2 || 3 && 3 > 2;
    printf("%d", result);

    result = x = y = 1;
    printf("%d", result);

    result = - ++x + y--;
    printf("%d", result);

    return 0;
}
```



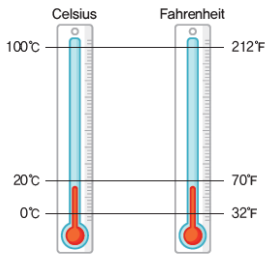
중간 점검

1. 연산자 중에서 가장 우선 순위가 낮은 연산자는 무엇인가?
2. 논리 연산자인 &&과 || 중에서 우선 순위가 더 높은 연산자는 무엇인가?
3. 단항 연산자와 이항 연산자 중에서 어떤 연산자가 더 우선 순위가 높은가?
4. 관계 연산자와 산술 연산자 중에서 어떤 연산자가 더 우선 순위가 높은가?

mini project: 화씨 온도를 섭씨로 바꾸기

- 화씨 온도를 섭씨 온도로 바꾸는 프로그램을 작성하여 보자.

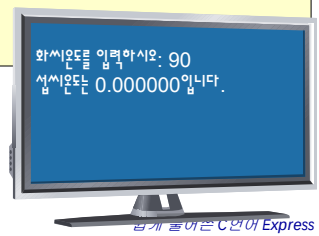
$$\text{섭씨온도} = \frac{5}{9}(\text{화씨온도} - 32)$$



잘못된 부분은 어디에?

```
#include <stdio.h>
int main(void)
{
        double f_temp;
        double c_temp;

        printf("화씨온도를 입력하시오");
        scanf("%lf", &f_temp);
        c_temp = 5 / 9 * (f_temp - 32);
        printf("섭씨온도는 %f입니다", c_temp);

        return 0;
}
```

c_temp = 5.0 / 9.0 * (f_temp - 32);



도전문제

- 위에서 제시한 방법 외에 다른 방법은 없을까?
- ((double)5 /(double)9) * (f_temp - 32); 가 되는지 확인하여 보자.
- ((double)5 /9) * (f_temp - 32); 가 되는지 확인하여 보자.

Q & A