

# 제 4장 변수와 자료형

# 이번 장에서 학습할 내용



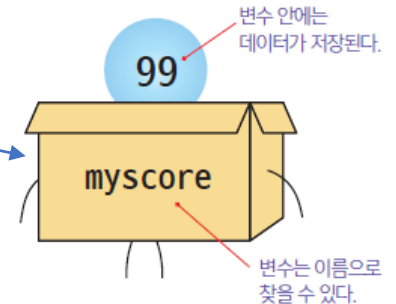
- \* 변수와 상수의 개념 이해
- \* 자료형
- \* 정수형
- \* 실수형
- \* 문자형
- \* 기호 상수 사용
- \* 오버플로우와 언더플로우 이해

이번 장에서는  
변수와 각종  
자료형을  
살펴봅니다.



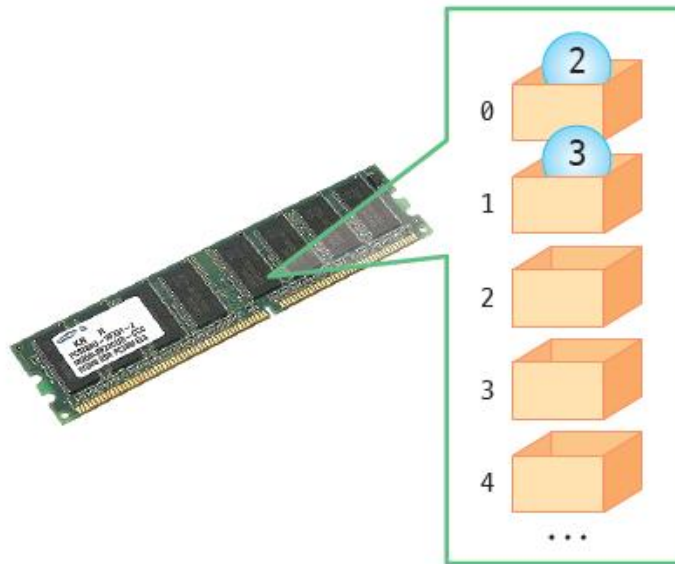
# 변수

- 컴퓨터 프로그램은 값을 저장하기 위하여 변수(variable)을 사용한다.
- 변수는 게임에서 점수를 저장하는데 사용될 수 있고, 대형 마트에서 우리가 구입한 물건들의 가격을 저장할 :



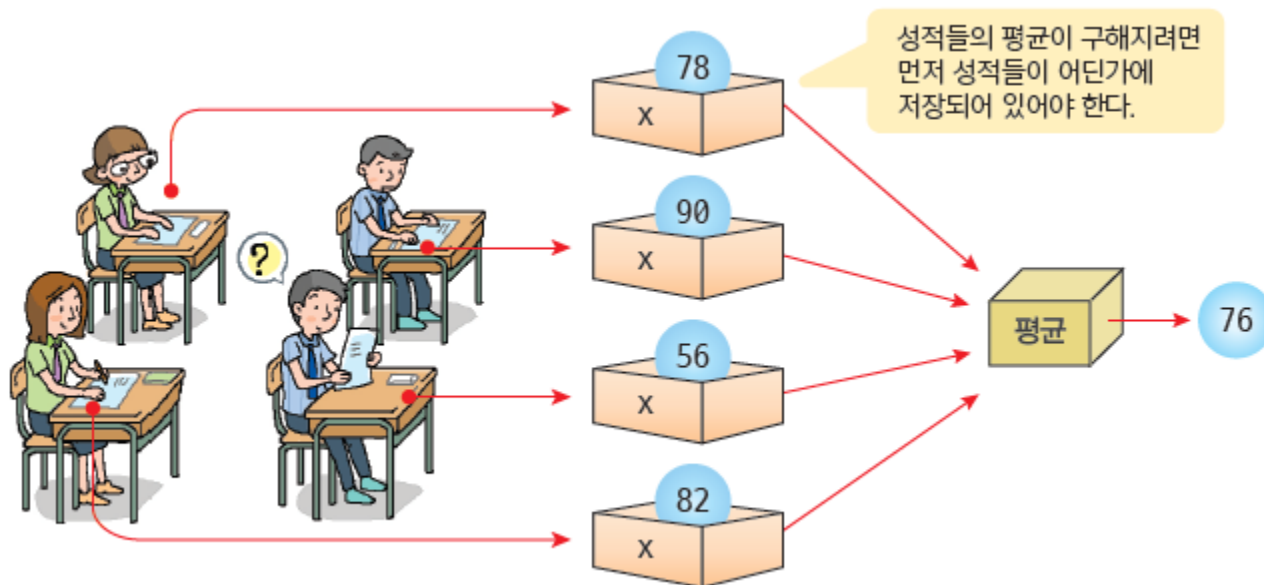
# 변수는 어디에 만들어지는가?

- 변수는 바로 메인 메모리(main memory)에 만들어진다.
- 우리는 변수 이름을 사용하여 메모리 공간을 사용하게 된다



# 변수가 왜 필요한가? #1

- 사용자에게서 받는 데이터를 저장하는 장소이다. – 변수가 없다면 사용자로부터 받은 데이터를 어디에 저장할 것인가?



# 변수가 왜 필요한가? #2

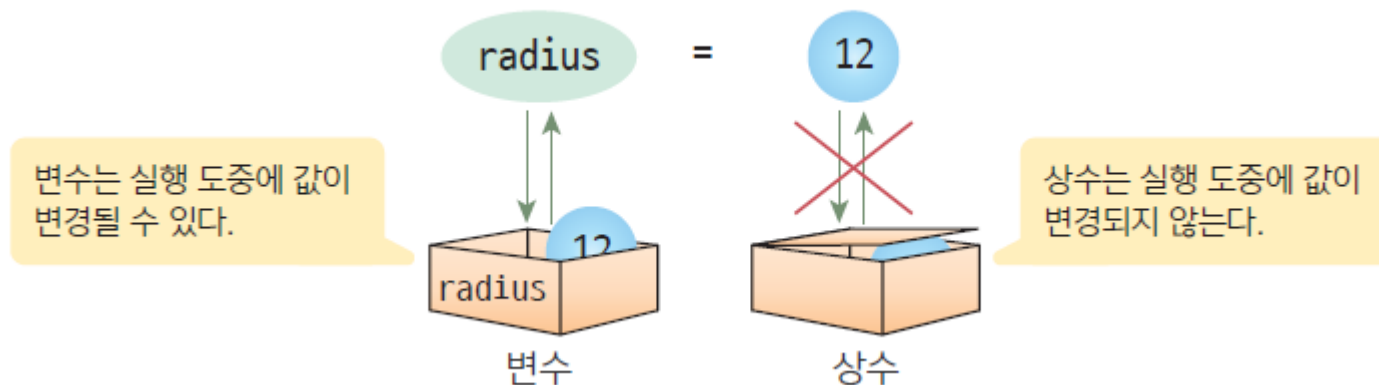
변수를 사용하지 않는 코드	변수를 사용하는 코드
<pre>// 크기가 100×200인 사각형의 면적 area = 100 * 200;</pre>	<pre>// 크기가 width×height인 사각형의 면적 width = 100; height = 200; area = width * height;</pre>



어떤 코드가 더  
유연한가요?  
변경에 더 잘 적응할 수  
있나요?

# 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;  
    float area;  
  
    printf("원의 면적을 입력하세요:");  
    scanf("%f", &radius);  
  
    area = 3.141592 * radius * radius;  
    printf("원의 면적: %f Wn", area);  
  
    return 0;  
}
```

변수

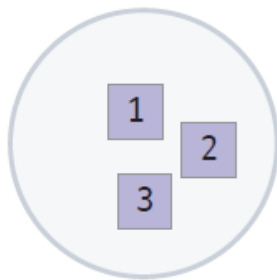
// 원의 반지름  
// 원의 면적

상수

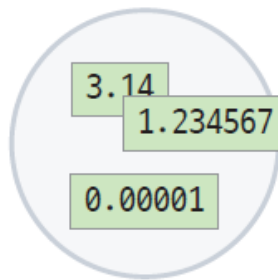


# 자료형

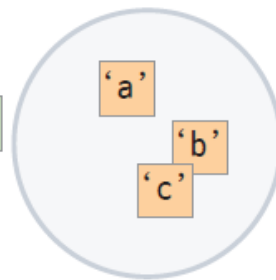
- 자료형(data type): 데이터의 타입(종류)
  - short, int, long: 정수형 데이터(100)
  - double, float: 부동소수점형 데이터(3.141592)
  - char: 문자형 데이터('A', 'a', '한')



정수형



부동 소수점형



문자형



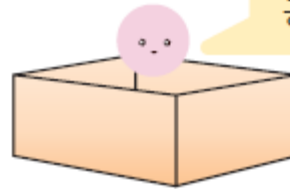
# 다양한 자료형이 필요한 이유

- 상자에 물건을 저장하는 것과 같다.

물건이 상자보다 크면  
들어가지 않을 것이다.

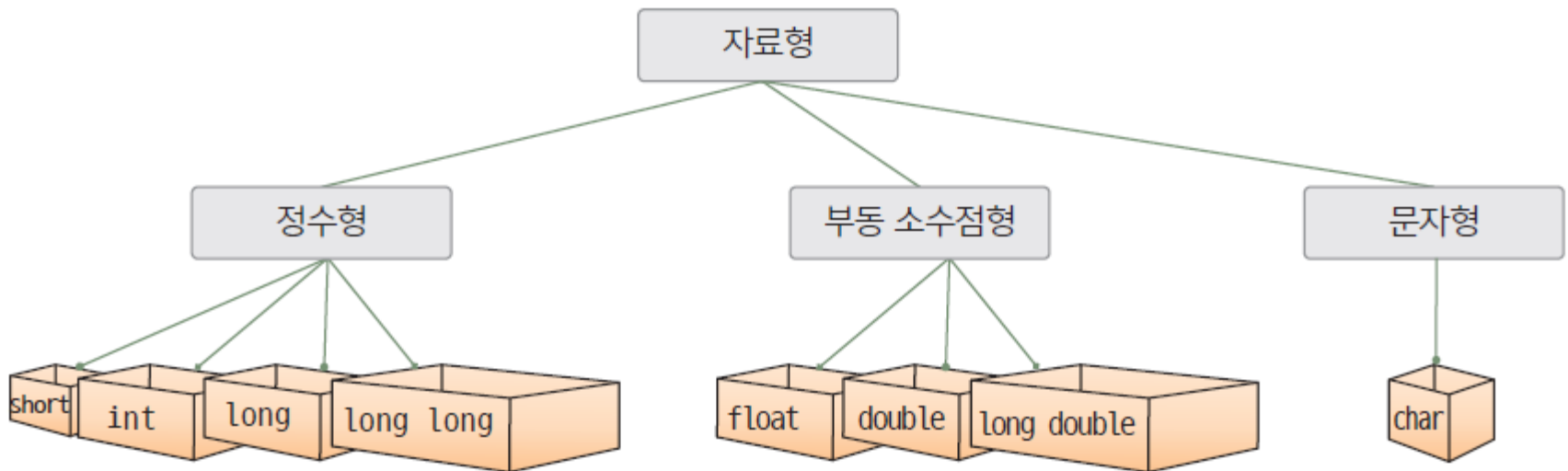


물건이 상자보다 너무 작으면  
공간이 낭비될 것이다.



# 자료형의 분류

- 자료형을 크게 나누면 정수형(integer type), 부동소수점형(floating-point type), 문자형(character type)으로 나눌 수 있다.



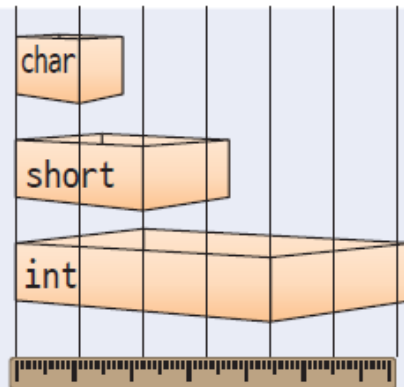
# 자료형의 크기

- 자료형의 크기를 알아보려면 sizeof 연산자를 사용하면 된다. sizeof는 변수나 자료형의 크기를 바이트 단위로 반환하는 연산자이다.

## Syntax sizeof()

예

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



sizeof 연산자는 변수나 자료형의 크기를 바이트 단위로 반환합니다.



# 예제: 자료형의 크기

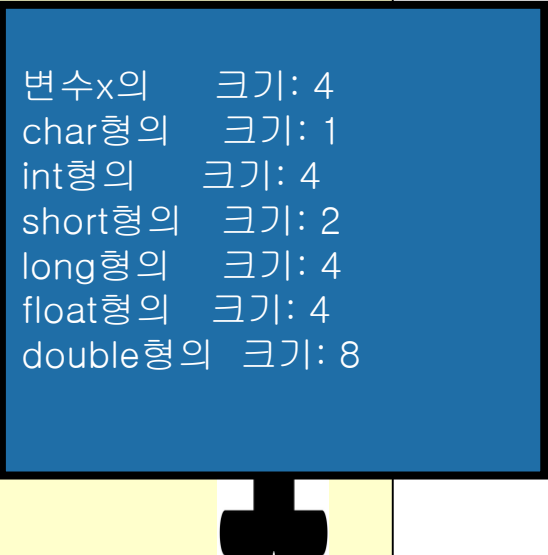
```
#include <stdio.h>

int main(void)
{
    int x;

    printf("변수x의   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("float형의  크기: %d\n", sizeof(float));
    printf("double형의 크기: %d\n", sizeof(double));

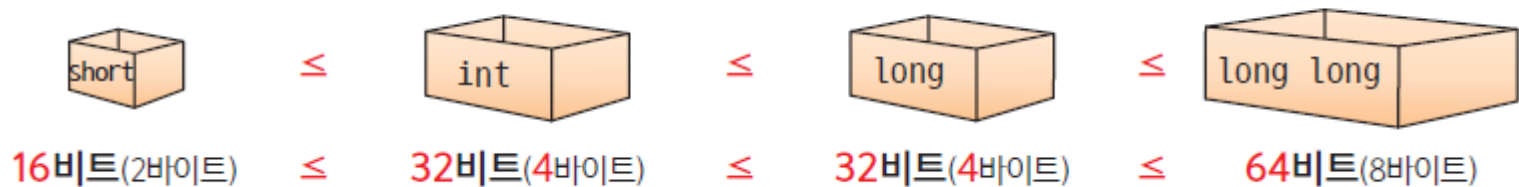
    return 0;
}
```



변수x의	크기: 4
char형의	크기: 1
int형의	크기: 4
short형의	크기: 2
long형의	크기: 4
float형의	크기: 4
double형의	크기: 8

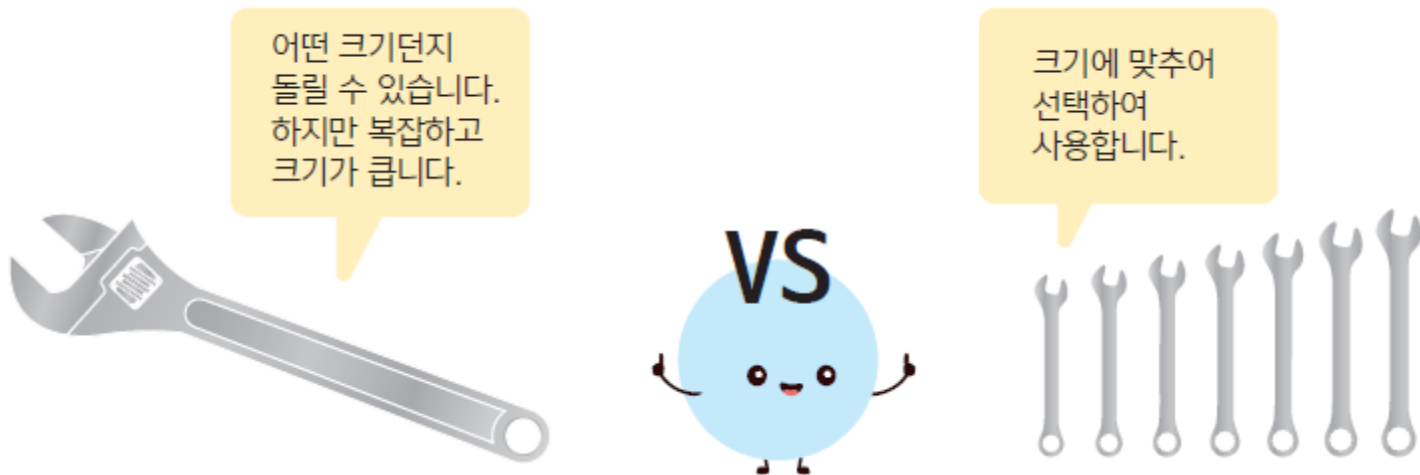
# 정수형

자료형		비트	범위
정수형	short	16비트	-32768~32767
	int	32비트	-2147483648~2147483647
	long		
	long long	64비트	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807



# C에서는 왜 이렇게 많은 종류의 정수형이 있을까?

- 용도에 따라 프로그래머가 선택하여 사용할 수 있게 하려는 것이다.
- 비트수를 늘리면 정수의 범위는 확대시킬 수 있지만 메모리 공간을 더 많이 필요로 한다.



# 정수형의 범위

- int형

$-2^{31}, \dots, -2, -1, 0, 1, 2, \dots, 2^{31} - 1$   
(-2147483648 ~ +2147483647)

- short형

$-2^{15}, \dots, -2, -1, 0, 1, 2, \dots, 2^{15} - 1$   
(-32768 ~ +32767)

- long형

- 보통 int형과 같음

약 -21억에서  
+21억





# 예제

```
/* 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;    // 64비트 자료형
```

```
    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;
```

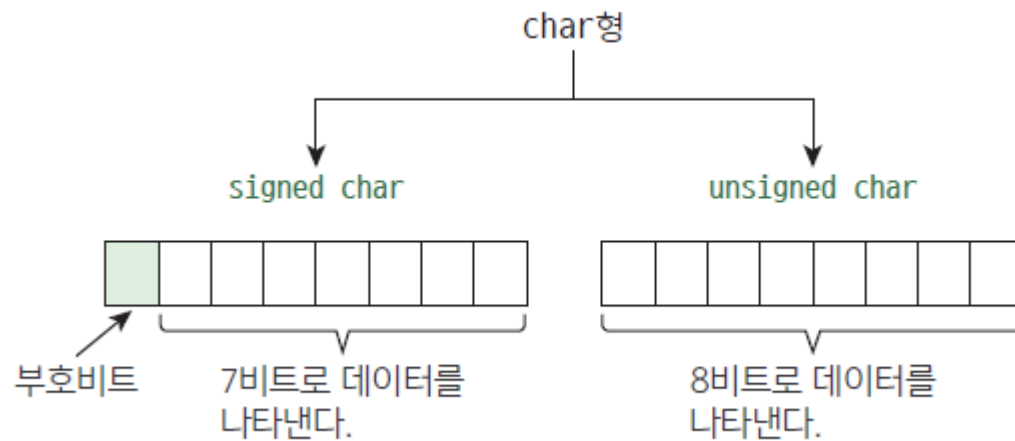
```
}
```



```
total_sale = 200000000
```

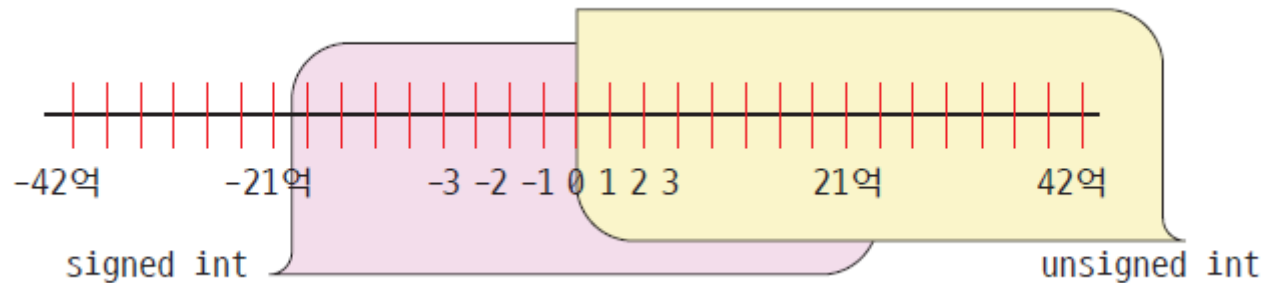
# signed, unsigned 수식자

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



# unsigned int

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



# unsigned 예제

```
unsigned int  speed;           // 부호없는 int형  
unsigned      distance;        // unsigned int distance와 같다.  
unsigned short players;        // 부호없는 short형
```

```
unsigned int sales = 2800000000; // 약 28억  
printf("%u \n", sales);          // %d를 사용하면 음수로 출력된다
```

unsigned는 %u로  
출력하세요.



# 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

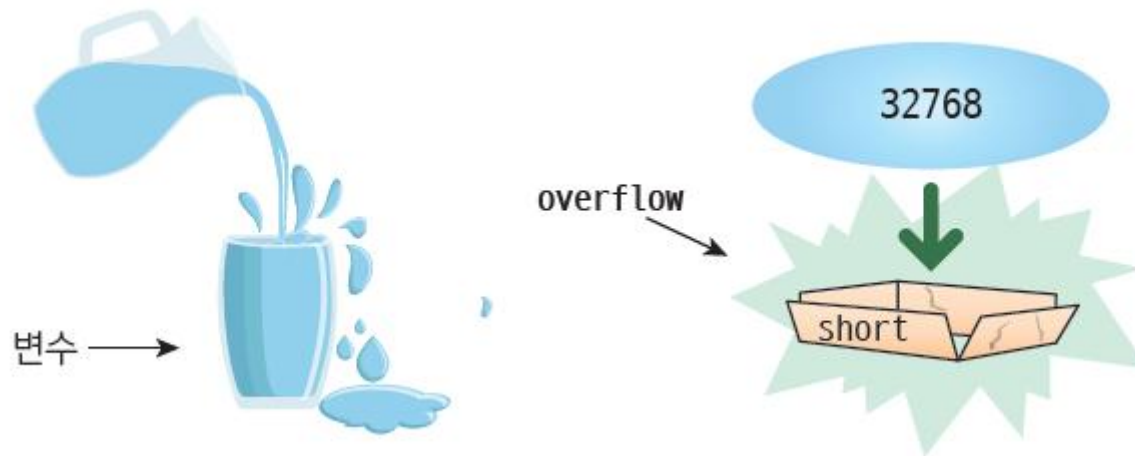
    s_money = s_money + 1;
    printf("s_money = %d", s_money);
    u_money = u_money + 1;
    printf("u_money = %d", u_money);
    return 0;
}
```

오버플로우 발생!!

S\_money = -32768  
U\_money = 0

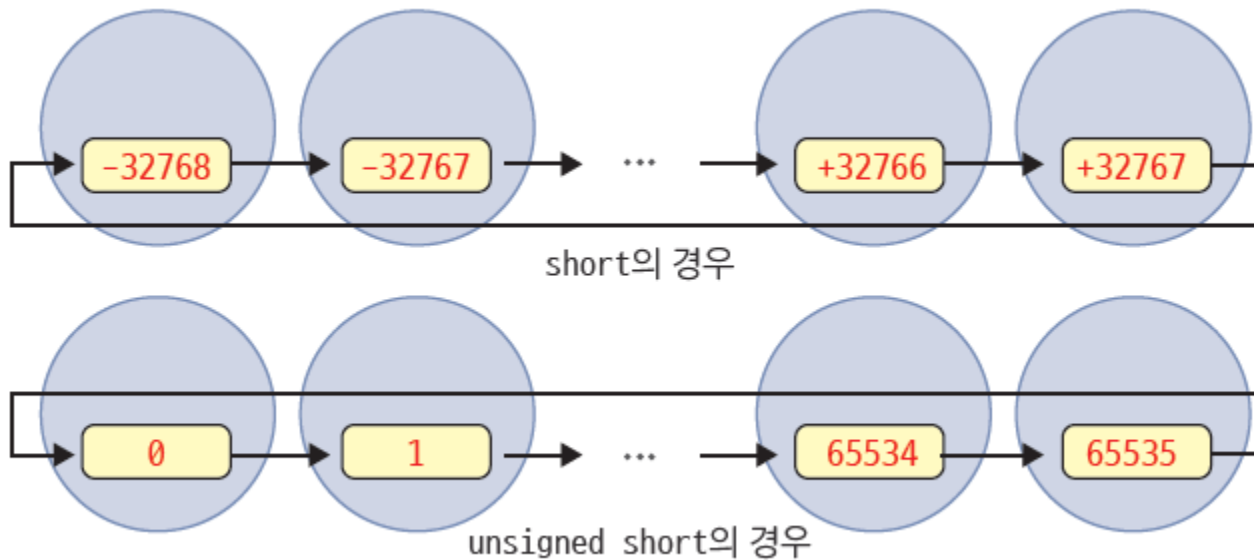
# 오버플로우

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



# 오버플로우

- 규칙성이 있다.
  - 수도 계량기나 자동차의 주행거리계와 비슷하게 동작



# 참고

## 참고사항

각 자료형의 최대값과 최소값은 limits.h에 정의되어 있다.

```
#define CHAR_MIN    (-128)
#define CHAR_MAX    127

#define SHRT_MIN    (-32768)    /* minimum (signed) short value */
#define SHRT_MAX    32767      /* maximum (signed) short value */
#define USHRT_MAX   0xffff     /* maximum unsigned short value */

#define INT_MIN     (-2147483647 - 1) /* minimum (signed) int value */
#define INT_MAX     2147483647      /* maximum (signed) int value */
#define UINT_MAX    0xffffffff     /* maximum unsigned int value */
```



# 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

```
/* 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;  
}
```

x = 10  
y = 8

x = 10  
y = 8  
z = 16

# 기호 상수

- 기호 상수(symbolic constant): 기호를 이용하여 상수를 표현한 것
- (예)
  - `won = 1120 * dollar;` // (1) 실제의 값을 사용
  - `won = EXCHANGE_RATE * dollar;` // (2) 기호상수 사용
- 기호 상수의 장점
  - 가독성이 높아진다.
  - 값을 쉽게 변경할 수 있다.

# 기호 상수의 장점

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

```
int main(void)
```

```
{
```

```
...
```

```
won1 = 1120 * dollar1;
```

```
won2 = 1120 * dollar2;
```

```
...
```

```
}
```

1050

1050

리터럴 상수를 사용하는 경우:  
등장하는 모든 곳을 수정하여야 한다.

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

```
#define EXCHANGE_RATE 1120
```

```
int main(void)
```

```
{
```

```
...
```

```
won1 = EXCHANGE_RATE * dollar1;
```

```
won2 = EXCHANGE_RATE * dollar2;
```

```
...
```

```
}
```

1050

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

# 기호 상수를 만드는 방법 #1

Syntax

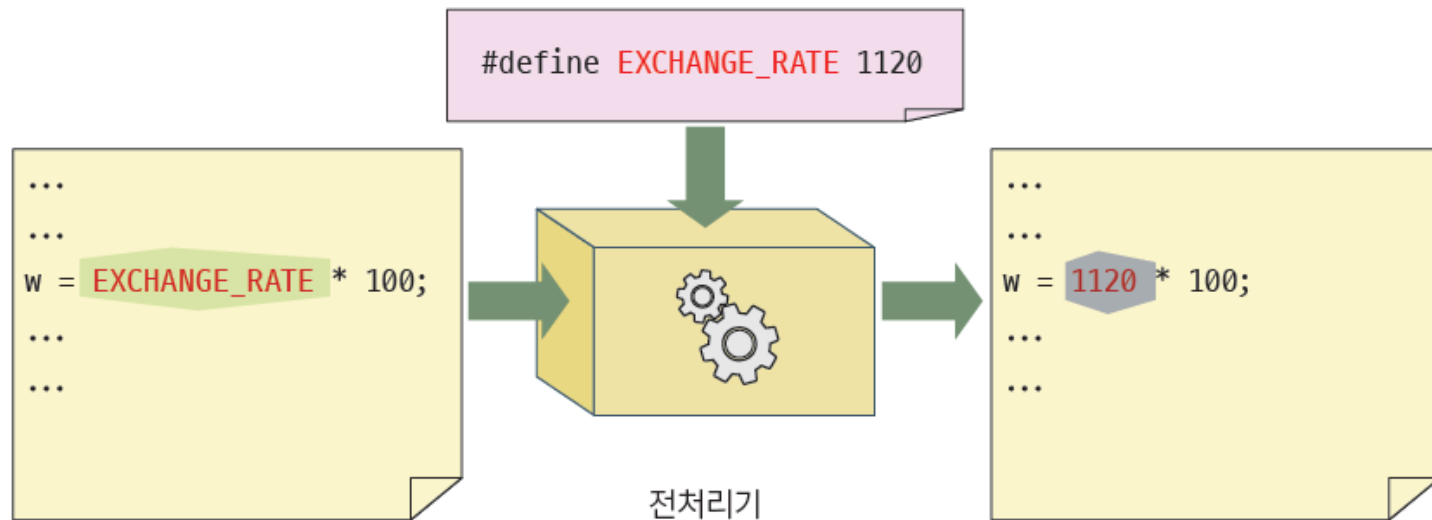
기호상수선언

예

```
#define EXCHANGE_RATE 1120
```

기호상수

값



# 기호 상수를 만드는 방법 #2

Syntax

기호상수선언

예

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

기호상수

값

# 예제: 기호 상수

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

```
#define TAX_RATE 0.2
```

기호상수

```
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;
```

```
}
```

월급을 입력하시요: 100  
연봉은 1200입니다.  
세금은 240.000000입니다.



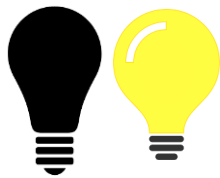
# 정수 표현 방법

- 컴퓨터에서 정수는 이진수 형태로 표현되고 이진수는 전자 스위치로 표현된다.

스위치가 **하나** 있으면



0, 1



스위치가 **둘** 있으면



00,



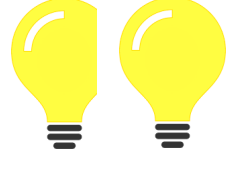
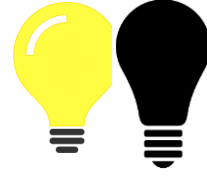
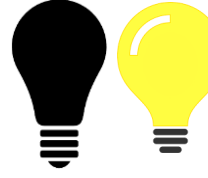
01,



10,



11

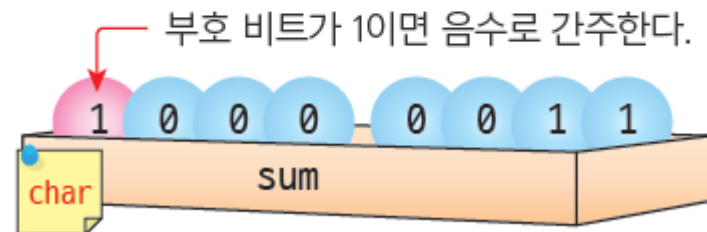
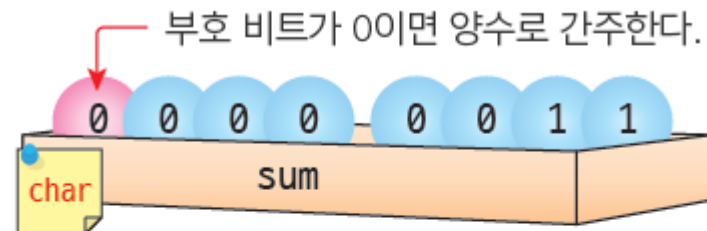


# short형

비트 패턴	정수	비고
0000000000000000	0	양의 정수
0000000000000001	1	
0000000000000010	2	
0000000000000011	3	
0000000000000100	4	
0000000000000101	5	
...	...	

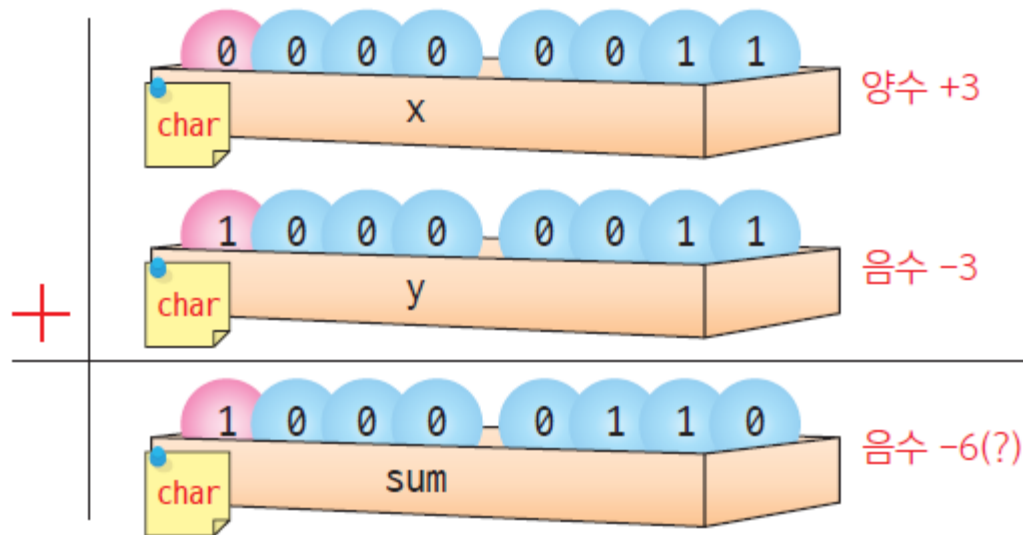
# 정수 표현 방법

- 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)  $+3 + (-3)$



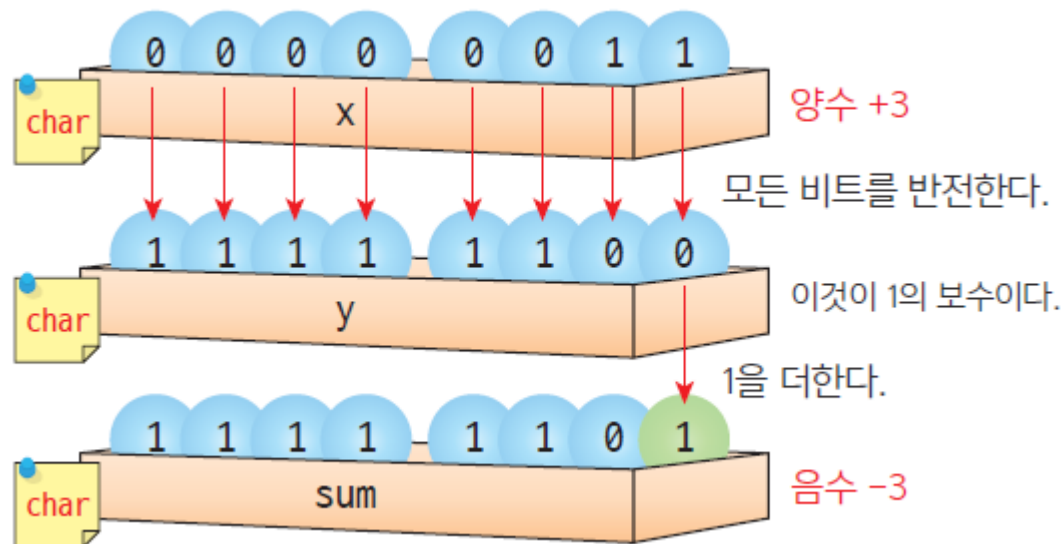
# 컴퓨터는 덧셈만 할 수 있다

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

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

# 음수를 표현하는 두번째 방법

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

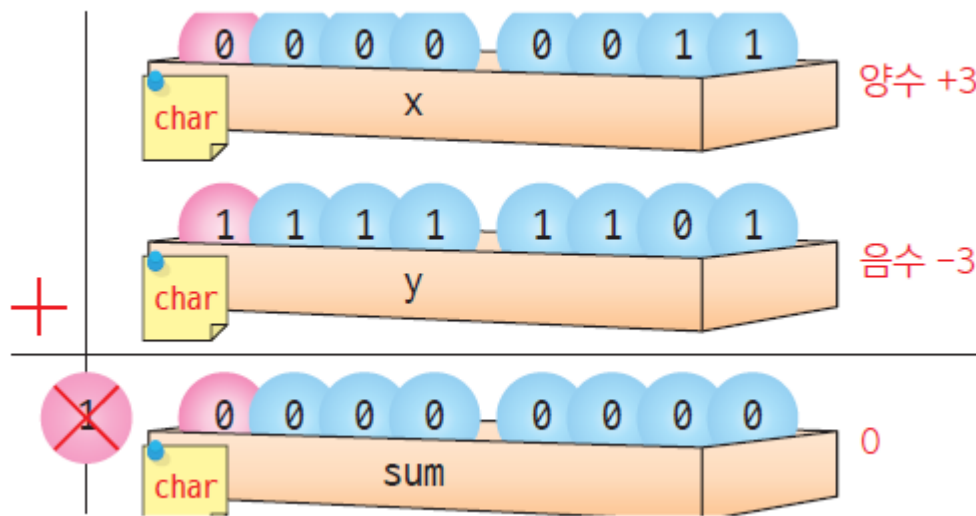


## 2의 보수

비트	부호없는 정수	부호있는 정수 (2의 보수)
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

비트	부호없는 정수	부호있는 정수 (2의 보수)
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
0111 1110	126	126
0111 1111	127	127
1000 0000	128	-128
1000 0001	129	-127
1000 0010	130	-126
1111 1110	254	-2
1111 1111	255	-1

# 2의 보수로 양수와 음수를 더하면



음수를 2의 보수로 표현하면 양수와 음수를 더할 때 각각의 비트들을 더하면 됩니다.





# 예제

```
/* 2의 보수 프로그램*/
```

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

```
int main(void)
```

```
{
```

```
    int x = 3;
```

```
    int y = -3;
```

```
    printf("x = %08X\n", x);
```

```
    printf("y = %08X\n", y);
```

```
    printf("x+y = %08X\n", x+y);
```

```
    return 0;
```

```
}
```

음수가 2의 보수로  
표현되는지를 알아보자.

// 8자리의 16진수로 출력한다.

// 8자리의 16진수로 출력한다.

// 8자리의 16진수로 출력한다.

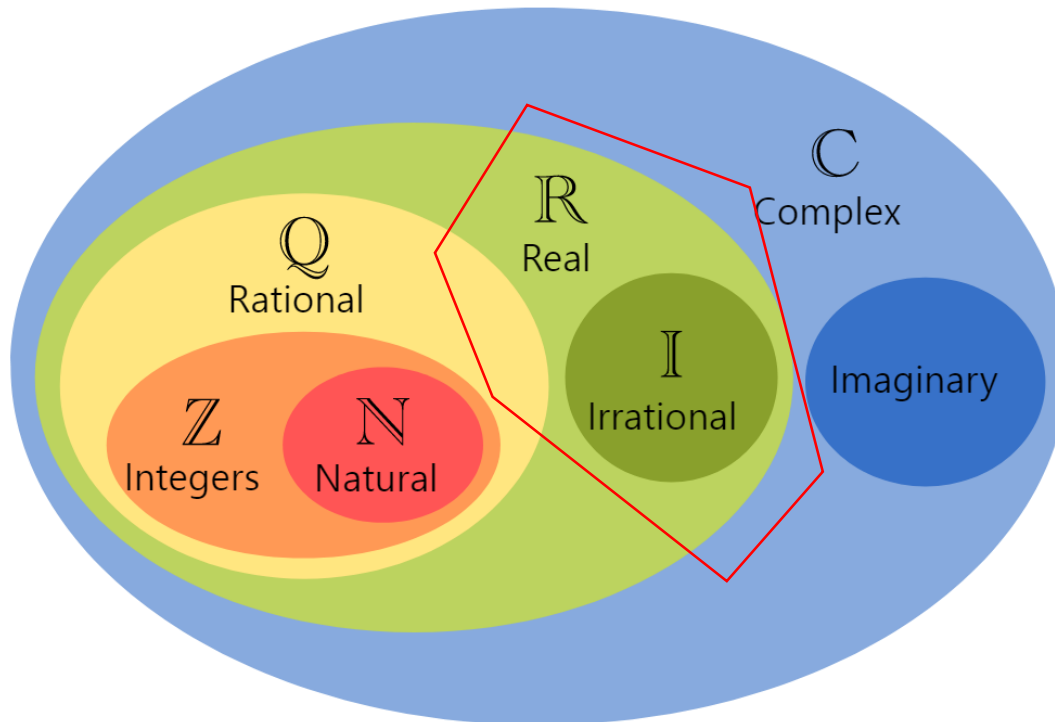
x = 00000003

y = FFFFFFFD

x+y = 00000000

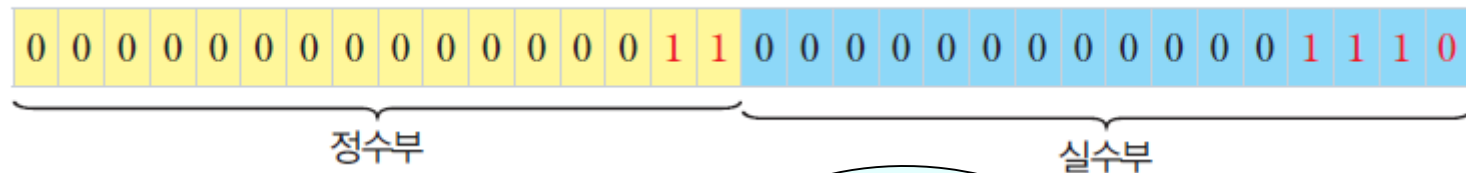
# 실수를 나타내는 방법

- 수학에서의 실수는 3.14와 같이 소수점을 가진 수이다. 실수는 매우 큰 수나 매우 작은 수를 다루는 과학이나 공학 분야의 응용 프로그램을 작성할 때는 없어서는 안 될 중요한 요소이다.



# 실수를 표현하는 방법 #1

- 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



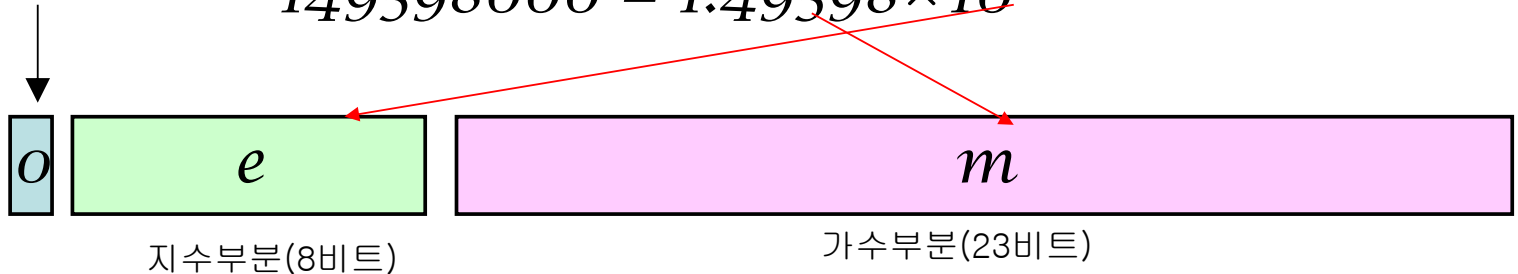
이 방법으로  
나타낼 수 있는  
최대수는  
얼마일까요?



# 실수를 표현하는 방법 #2

- 부동 소수점 방식

부호비트(1비트)



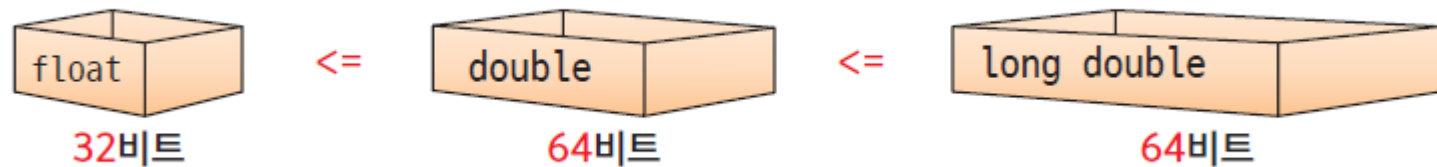
$$\text{실수값} = (-1)^s * (1.m) * 2^{e-127}$$

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

한 동안 아주 다양한  
부동 소수점 방법이  
사용 되었 지 만  
1985 년 부 터 는 IEEE  
754 로 표 준 화  
되었습니다.



# 부동 소수점 형



자료형	명칭	크기	범위
float	단일 정밀도(single-precision) 부동 소수점	32비트	$\pm 1.17549 \times 10^{-38} \sim \pm 3.40282 \times 10^{+38}$
double long double	두배 정밀도(double-precision) 부동 소수점	64비트	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{+308}$

# 실수를 출력하는 형식 지정자

- %f
  - `printf("%f", 0.123456789);` // 0.123457 출력
- %e
  - `printf("%e", 0.123456789);` // 1.234568e-001 출력

# 예제

```
/* 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	$1.2345 \times 10^2$
12345.0	1.2345e4	$1.2345 \times 10^4$
0.000023	2.3e-5	$2.3 \times 10^{-5}$
2,000,000,000	2.0e9	$2.0 \times 10^9$

1.23456

2.

.28

2e+10

9.26E3

0.67e-9

// 소수점만 붙여도 된다.

// 정수부가 없어도 된다.

// +나 -기호를 지수부에 붙일 수 있다.

//  $9.26 \times 10^3$

//  $0.67 \times 10^{-9}$



# 부동소수점 오버플로우

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

```
int main(void)
```

```
{
```

```
float x = 1e39;
```

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

```
}
```

숫자가 커서 오버플로우 발생

x = inf

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)
```

```
{
```

```
    float value = 0.1;
```

```
    printf("%.20f \n", value);
```

출력하라는 의미이다.

```
    return 0;
```

```
}
```

이진법으로는 정확하게 나타낼 수 없는 값들이 있기 때문이다. 0.1도 그 중의 하나이다.

// %.20f는 소수점 이하를 20자리로

0.10000000149011611938

# 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;
```

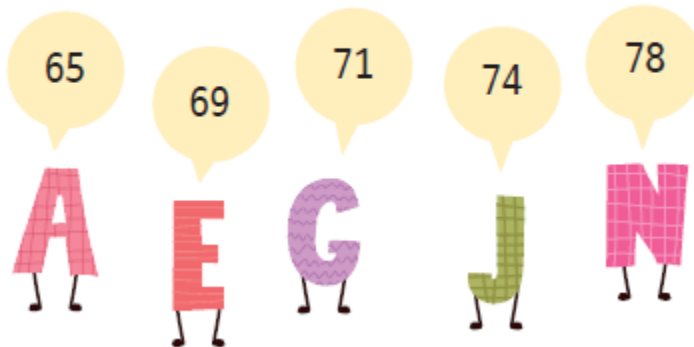
```
}
```

부동소수점 연산에서는  
오차가 발생한다.  
5.0이 아니라 0으로 계산  
된다.

0.000000

# 문자형

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



# 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

## 아스키 코드표 (일부)

Dec	Hex	문자
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c

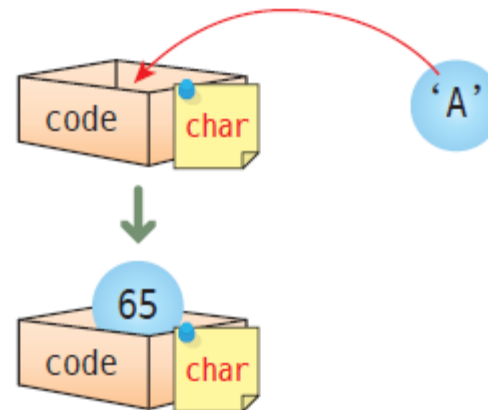
Dec	Hex	문자
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w

[illegible]

# 문자 변수

- char형을 사용하여 문자를 저장한다.

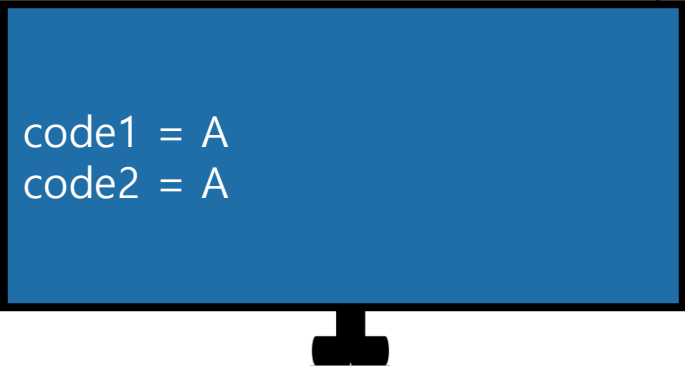
```
char code;  
code = 'A';
```





# 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
```

# Tip: 한글 표현 방법

## 한글 표현 방법

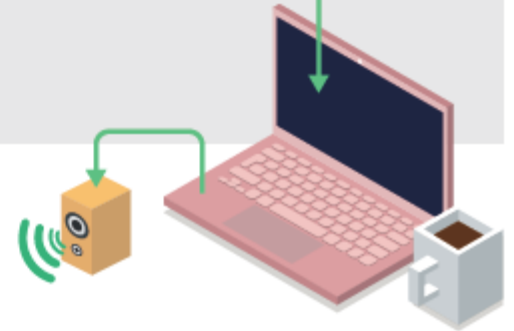
한글은 8비트로 표현할 수 없다. 글자의 개수가 영문자에 비하여 많기 때문이다. 컴퓨터에서 한글을 표현하는 방식은 크게 2가지 방법이 있다. 첫 번째는 이는 각 글자마다 하나의 코드를 부여하는 것이다. 예를 들어서 '가'에는 0xb0a1이라는 코드를 부여하는 것이다. 한글에서 표현 가능한 글자의 개수는 11172개이고 따라서 이들 글자에 코드를 부여하려면 8비트( $2^8=256$ )로는 부족하고, 16비트( $2^{16}=65536$ )가 되어야 가능하다. 이것을 완성형이라고 한다. 대표적인 코드 체계가 유니코드(unicode)이다. 유니코드(unicode)는 전세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이다. 유니코드 협회(unicode consortium)가 제정하며, 현재 최신판은 유니코드 10.0이다. 이 표준에는 문자 집합, 문자 인코딩, 문자 정보 데이터베이스, 문자들을 다루기 위한 알고리즘 등이 포함된다. 또 하나의 방법은 똑같이 16비트를 사용하는 방법이지만 글자의 초성, 중성, 종성에 각각 5비트씩을 할당하고, 가장 앞의 비트는 영숫자와 한글을 구분 짓는 기호로 하는 방법이다. 즉 맨 처음 비트가 1이면 한글, 0이면 영숫자로 판단하는 것이다. 이런 식으로 한글을 표현하는 방법을 조합형이라고 한다.

# 제어 문자

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

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

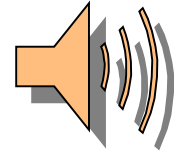
```
printf("%c", 7);
```



# 제어 문자를 나타내는 방법

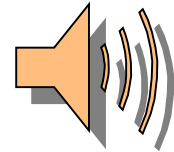
- 아스키 코드를 직접 사용

```
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 강제로 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

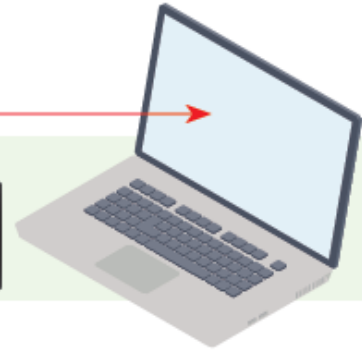
# 프로그램에서 경고음을 내려면?

특수 문자열을 사용하여 프로그램에서 경고음을 내려면 다음과 한다.

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

```
printf("\a");
```

beep



# 역슬래시 ₩

- 특수한 기능을 가진 문자 앞에 역슬래시 ₩를 위치시키면 문자의 특수한 의미가 사라진다.

```
printf("\"나만의 할리우드\" UCC 열풍 ");
```



“나만의 할리우드” UCC 열풍

```
printf("\\는 제어 문자를 표시할 때 사용한다. ");
```



\는 제어 문자를 표시할 때 사용한다.

# 예제

```
#include <stdio.h>
int main(void)
{
    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 4개의 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(void)
{
    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

# Lab: 변수의 초기값

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

    printf("3개의 정수를 입력하세요 (x, y, z): ");
    scanf("%d %d %d", &x, &y, &z);
    sum += x;
    sum += y;
    sum += z;
    printf("3개 정수의 합은 %d\n", sum);
    return 0;
}
```

## Microsoft Visual C++ Runtime Library



### Debug Error!

Program: ..\documents\visual studio  
2017\Projects\hello\Debug\hello.exe  
Module: ..\documents\visual studio  
2017\Projects\hello\Debug\hello.exe  
File:

Run-Time Check Failure #3 - The variable 'sum' is being used  
without being initialized.

(Press Retry to debug the application)

종단(A)

다시 시도(R)

무시(I)

# 무엇이 문제일까?

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

    sum = 0;
    printf("3개의 정수를 입력하세요 (x, y, z): ");
    scanf("%d %d %d", &x, &y, &z);
    sum += x;
    sum += y;
    sum += z;
    printf("3개 정수의 합은 %d\n", sum);

    return 0;
}
```

변수는 사용하기 전에  
반드시 초기화 시켜야 함!

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

# Q & A

