

프로그래밍 연습 Lab 2

# C 프로그램 변수와 자료형

---

[TA] 강성민, 김기현, 최석원, 최지은, 표지원  
Department of Computer Science and Engineering  
Seoul National University, Korea  
2022/09/21

# 이번 장에서 학습할 내용

---

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

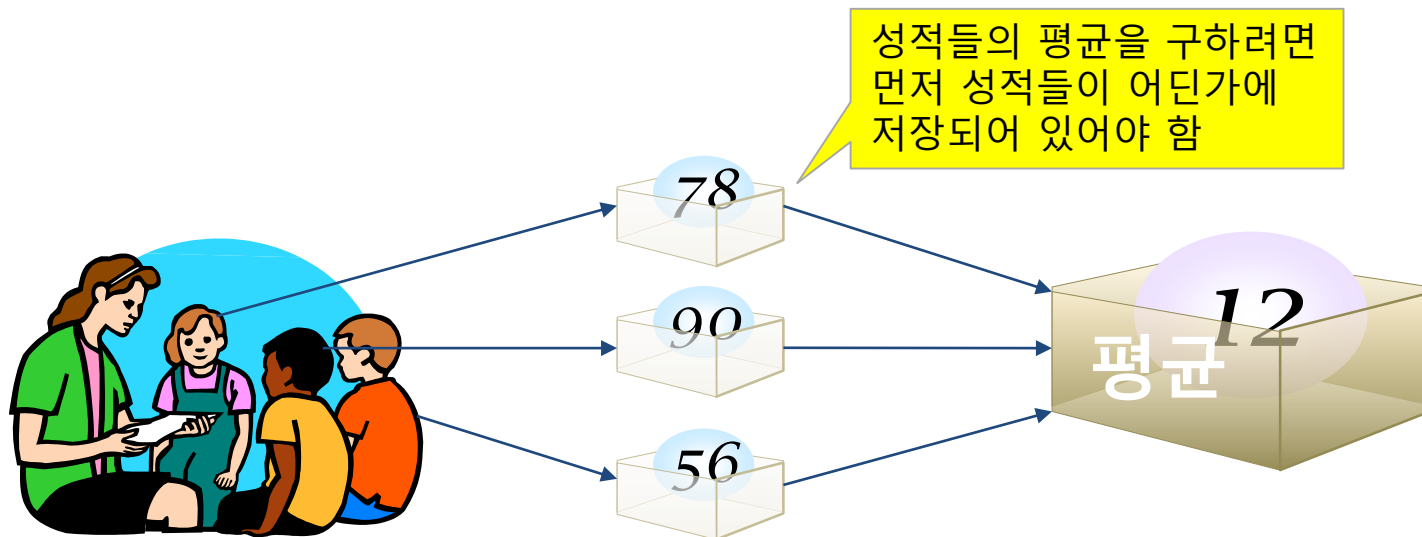
# 변수

Q) 변수(variable)이란 무엇인가?

A) 프로그램에서 일시적으로 데이터를 저장하는 공간

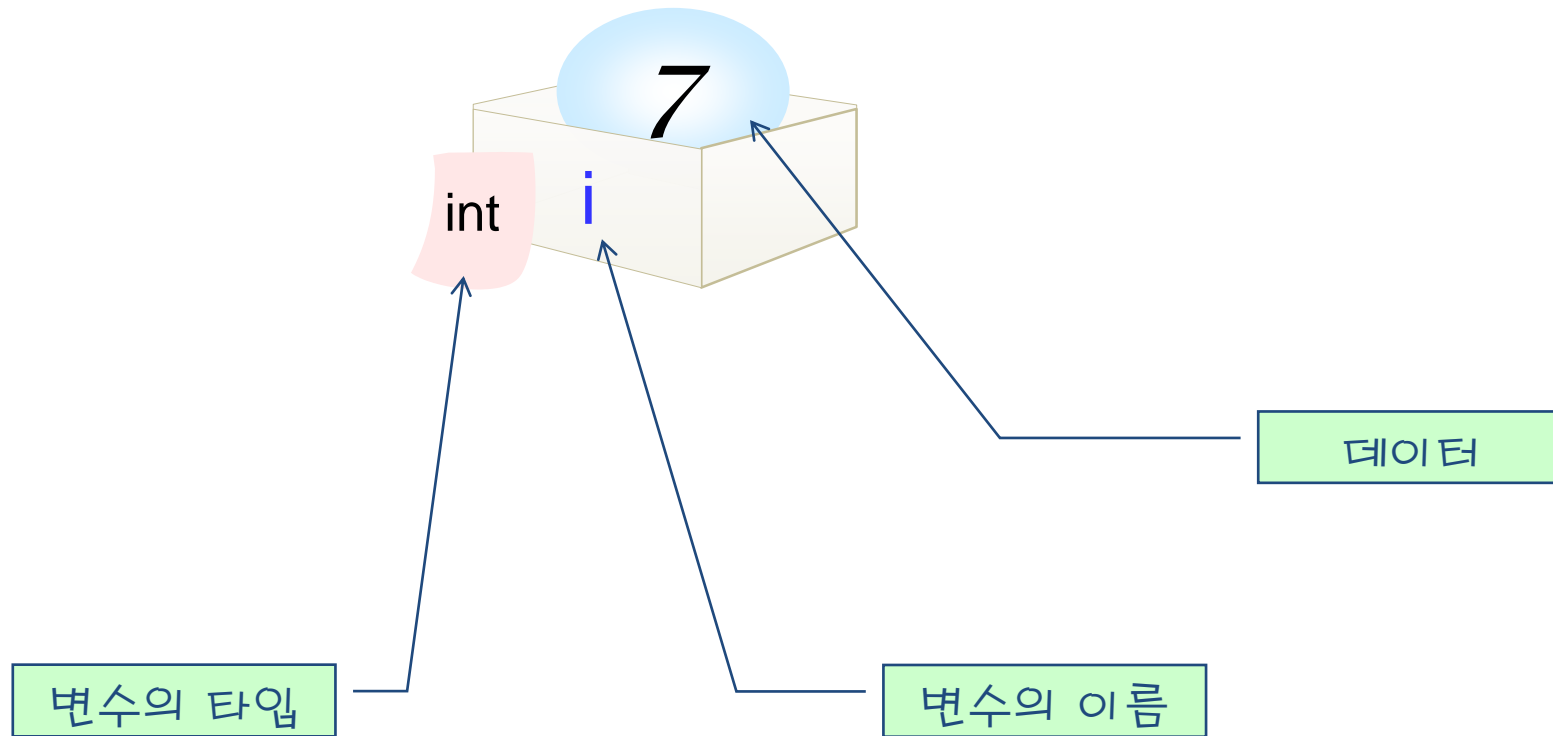
Q) 변수는 왜 필요한가?

A) 데이터가 입력되면 어딘가에 저장해야만 다음에 사용할 수 있다.



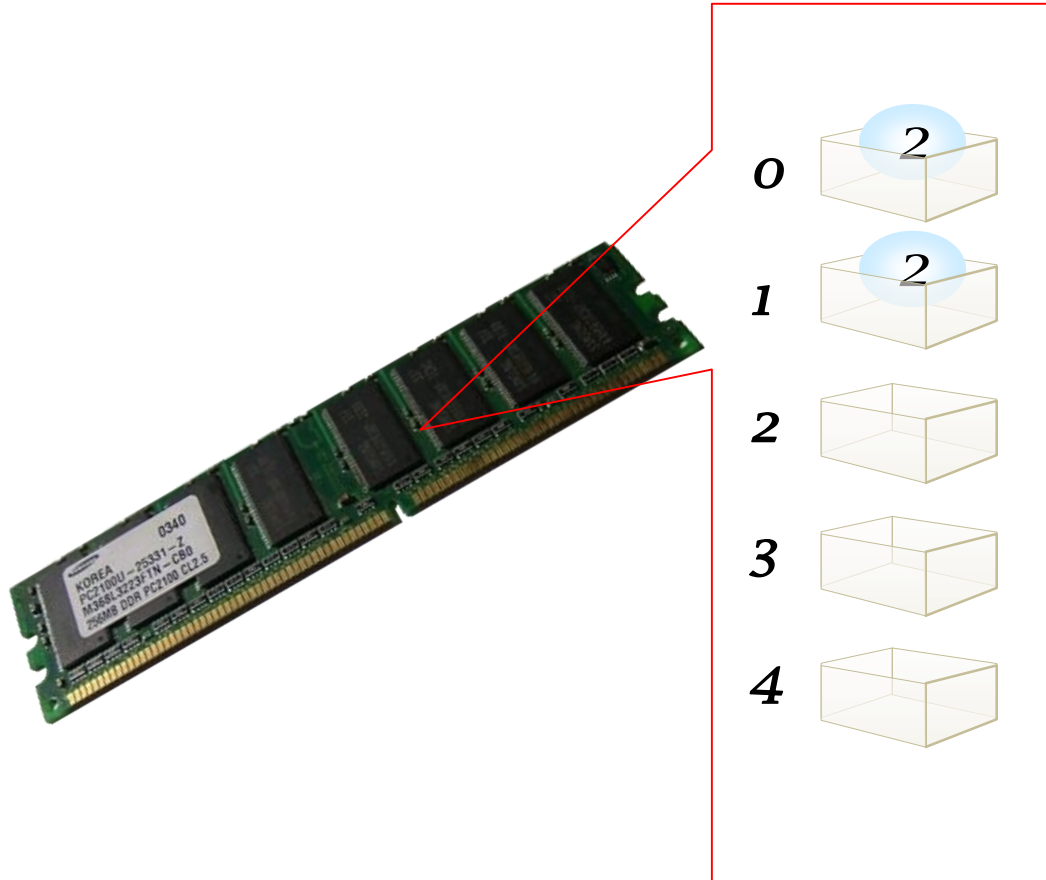
# 변수 = 상자

- 변수는 물건을 저장하는 상자와 같다.



# 변수가 만들어지는 곳

- 변수는 메인 메모리에 만들어진다.

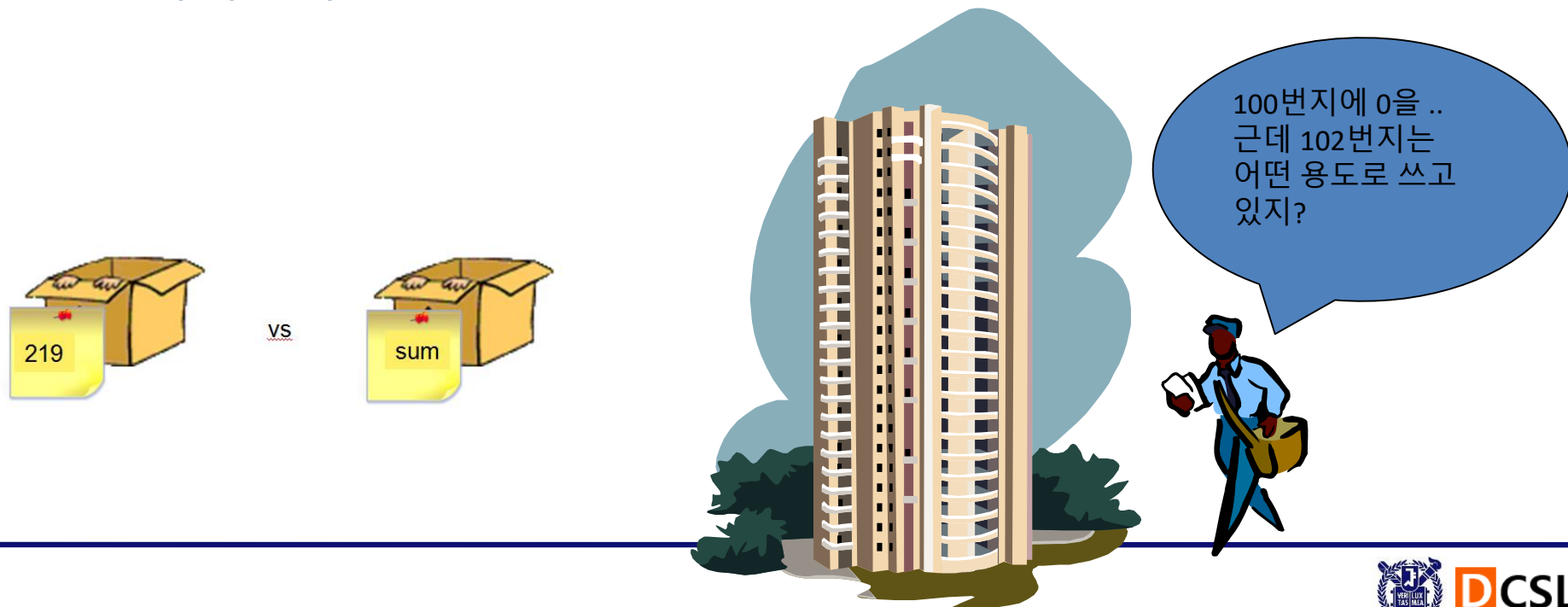


# 메모리를 주소로 사용한다면

(Q) 만약 메모리를 변수처럼 이름을 가지고 사용하자 않고 주소로 사용한다면?

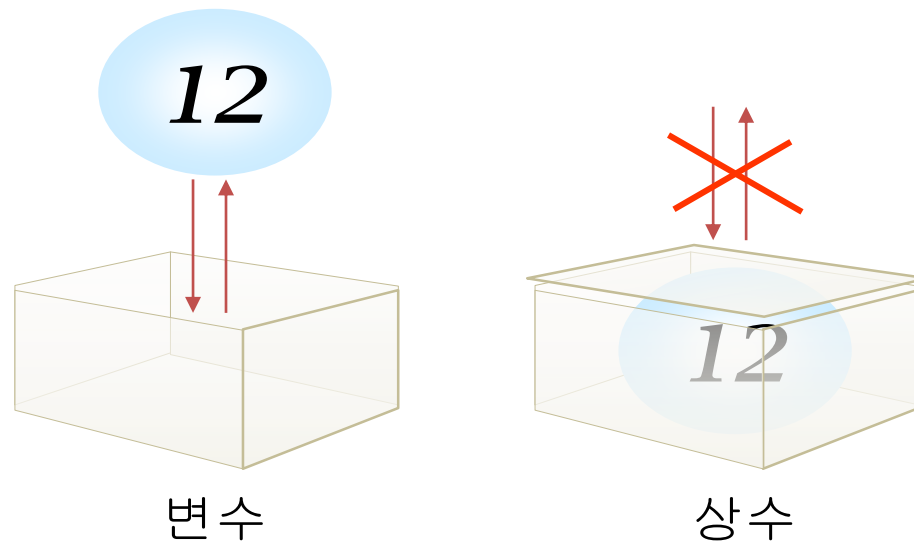
“100번지에 0을 대입하라”

(A) 충분히 가능하지만 불편하다. 인간은 숫자보다는 기호를 더 잘 기억한다.



# 변수와 상수

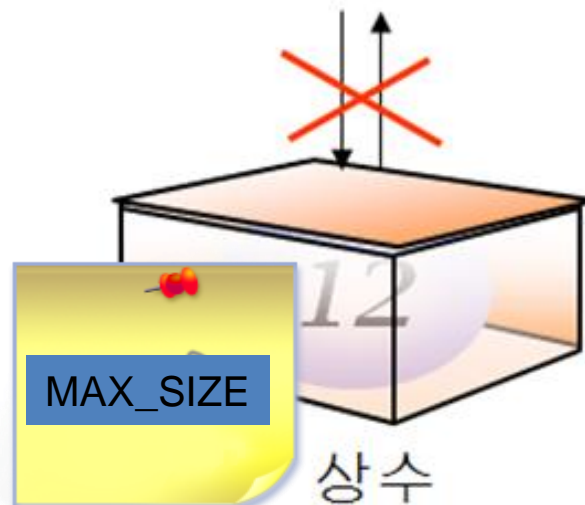
- **변수(variable)**: 저장된 값의 변경이 가능한 공간
- **상수(constant)**: 저장된 값의 변경이 불가능한 공간  
(예) 3.14, 100, 'A', "Hello World!"



# 상수의 이름

(Q) 상수도 이름을 가질 수 있는가?

(A) 보통 상수는 이름이 없다. 이러한 상수를 리터럴(literal)이라고 한다. 하지만 필요하다면 상수에도 이름을 붙일 수 있다. 이것을 기호 상수라고 한다.





# 예제: 변수와 상수

```
/* 원의 면적을 계산하는 프로그램 */
```

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

```
int main(void)
```

```
{
```

```
float radius;
```

```
float area;
```

// 원의 반지름

// 원의 면적

```
printf("원의 면적을 입력하세요:");
```

```
scanf("%f", &radius);
```

```
area = 3.141592 * radius * radius;
```

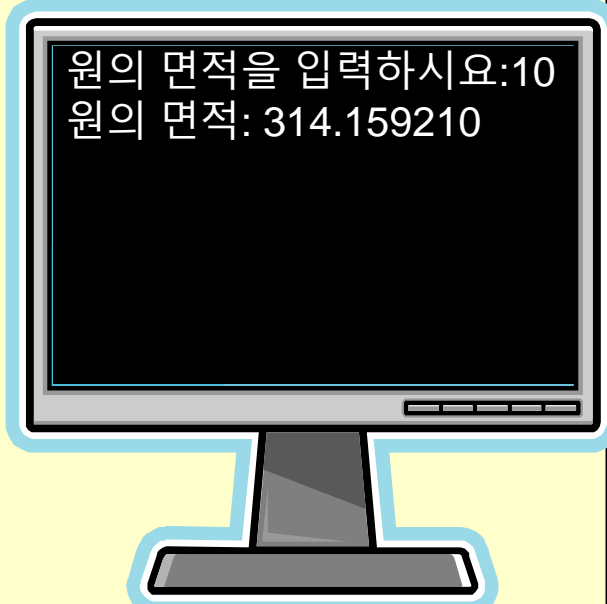
```
printf("원의 면적: %f\n", area);
```

```
return 0;
```

```
}
```

변수

상수



원의 면적을 입력하세요:10  
원의 면적: 314.159210

# 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')



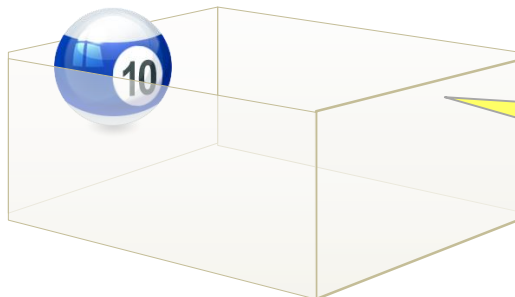
# 다양한 자료형이 필요한 이유

(Q) 다양한 자료형이 필요한 이유는?

(A) 상자에 물건을 저장하는 것과 같다.

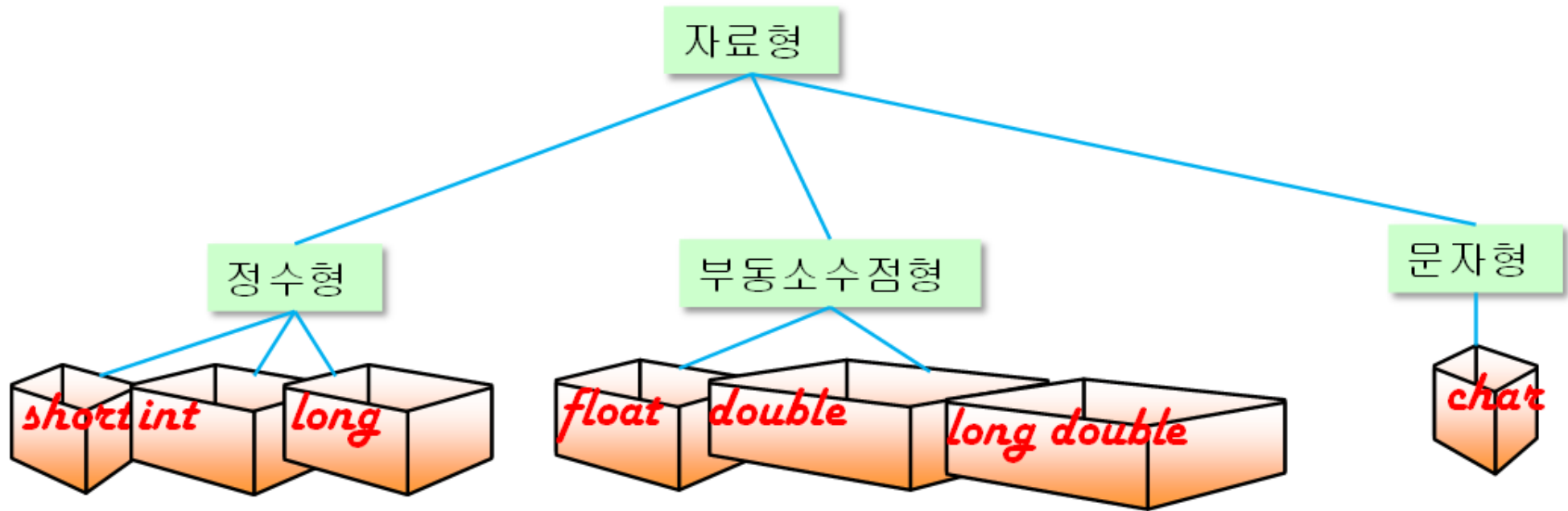


물건이 상자보다 크면 들어가지 않을 것이다.

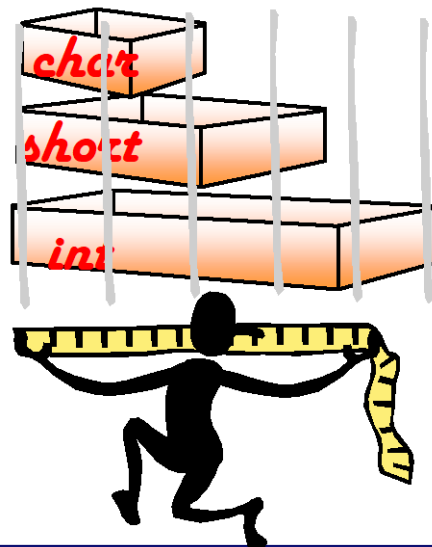
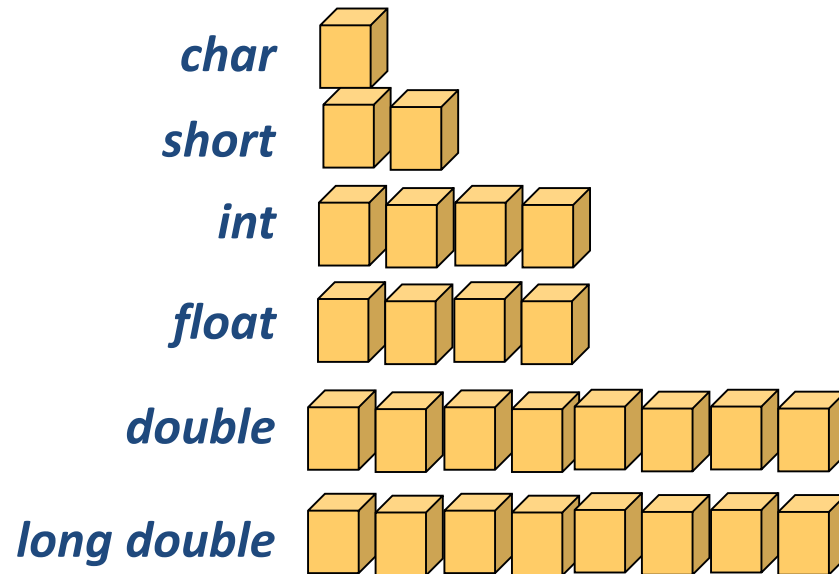


물건이 상자보다 너무 작으면 공간이 낭비될 것이다.

# 자료형



# 자료형의 크기



sizeof 연산자는 변수나 데이터 타입의 크기를 바이트 단위로 반환합니다.

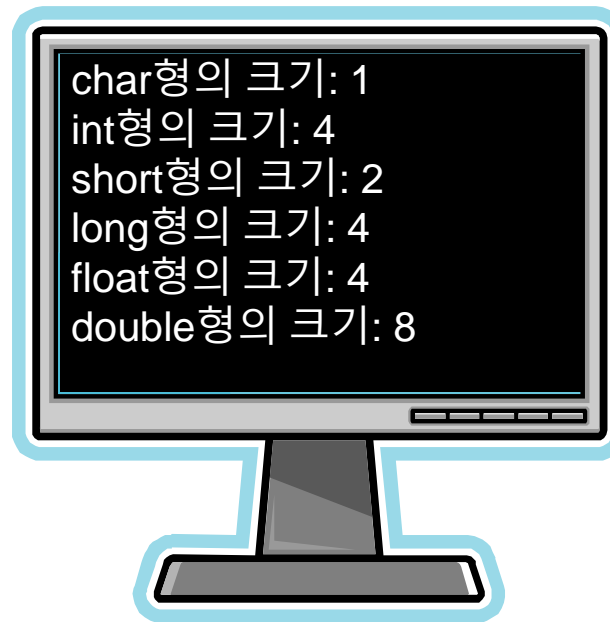


# 자료형의 크기 출력

---

`sizeof()` 함수를 사용하여 `sizeof` 함수 안에 자료형을 넣어 자료의 크기(byte)를 알 수 있음  
ex) `sizeof(int)`

Int, short, long, float double 자료형의 크기가 몇바이트인지 출력 해보기

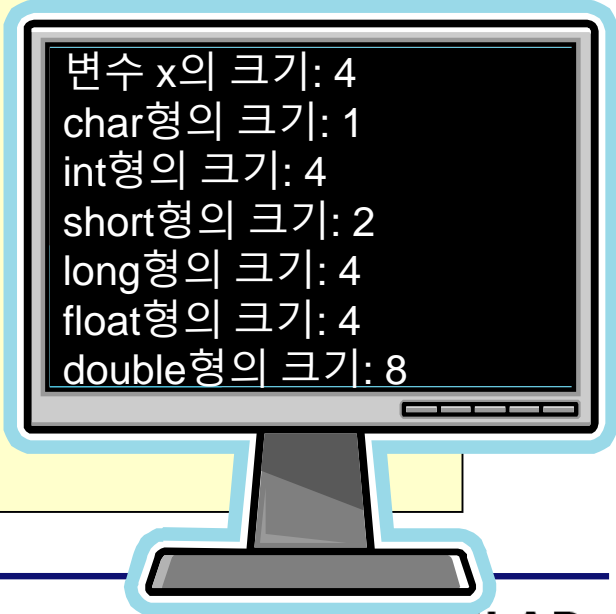


# 예제: 자료형의 크기

```
#include <stdio.h>

int main(void)
{
    int x;

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



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

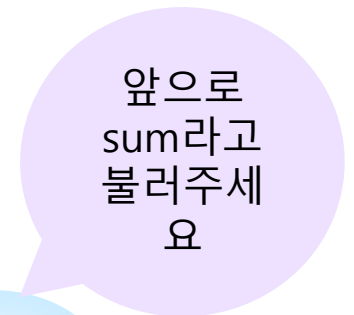
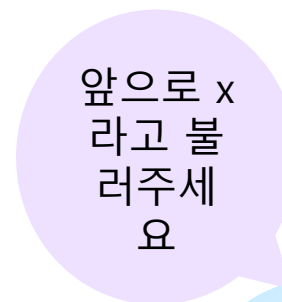
# 자료형의 종류

자료형			설명	바이트수	범위
정수형	부호있음	short	short형 정수	2	-32768 ~ 32767
		int	정수	4	-2147483648 ~ 2147483647
		long	long형 정수	4	-2147483648 ~ 2147483647
	부호없음	unsigned short	부호없는 short형 정수	2	0 ~ 65535
		unsigned int	부호없는 정수	4	0 ~ 4294967295
		unsigned long	부호없는 long형 정수	4	0 ~ 4294967295
문자형	부호있음	char	문자 및 정수	1	-128 ~ 127
	부호없음	unsigned char	문자 및 부호없는 정수	1	0 ~ 255
부동소수점형		float	단일정밀도 부동소수점	4	1.2E-38 ~ 3.4E38
		double	두배정밀도 부동소수점	8	2.2E-308 ~ 1.8E308
		long double	두배정밀도 부동소수점	8	2.2E-308 ~ 1.8E308



# 변수의 이름짓기

- 식별자(identifier): 식별할 수 있게 해주는 이름
  - 변수 이름
  - 함수 이름



# 식별자를 만드는 규칙

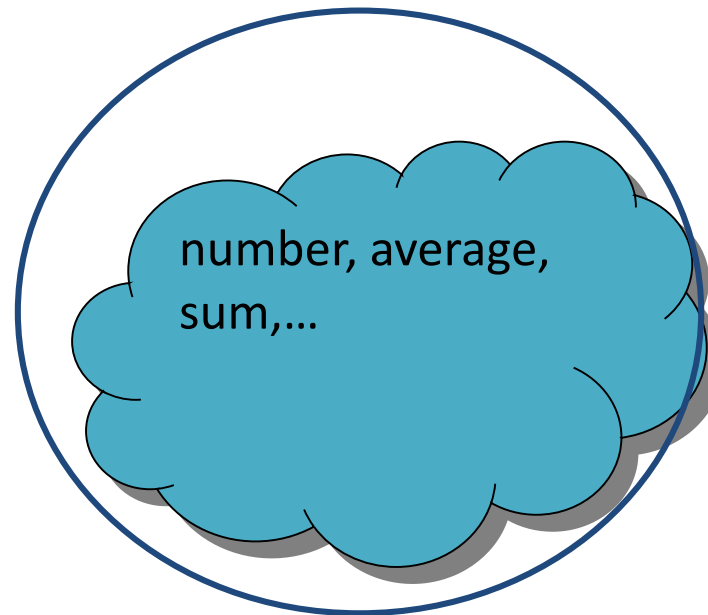
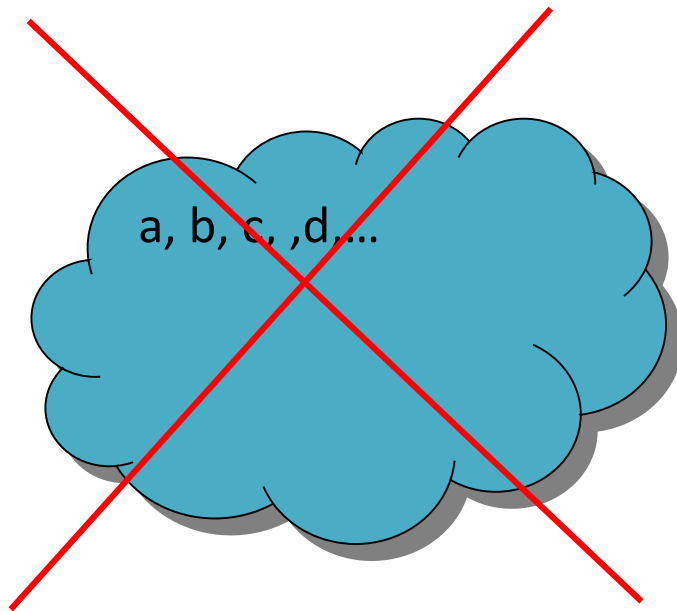
- 알파벳 문자와 숫자, 밑줄 문자 \_로 구성
- 첫 번째 문자는 반드시 알파벳 또는 밑줄 문자 \_
- 대문자와 소문자를 구별
- C 언어의 키워드와 똑같은 이름은 허용되지 않는다.

(Q) 다음은 유효한 식별자인가?

sum	O	
_count	O	
king3	O	
n_pictures	O	
2nd_try	X	// 숫자로 시작
Dollor#	X	// #기호
double	X	// 키워드

# 좋은 변수 이름

- 변수의 역할을 가장 잘 설명하는 이름
  - 밑줄 방식: `bank_account`
  - 단어의 첫번째 글자를 대문자: `BankAccount`



# 키워드

- 키워드(keyword): C언어에서 고유한 의미를 가지고 있는 특별한 단어
- 예약어(reserved words) 라고도 한다.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

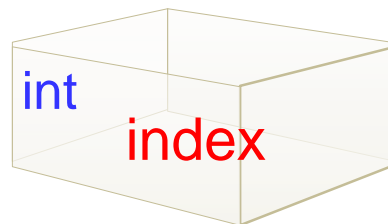
# 변수 선언

- 컴파일러에게 어떤 변수를 사용하겠다고 미리 알리는 것

자료형

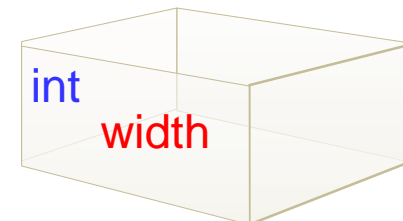
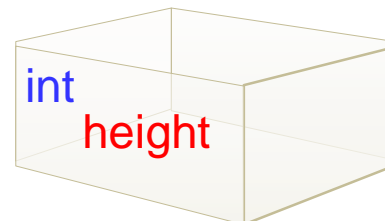
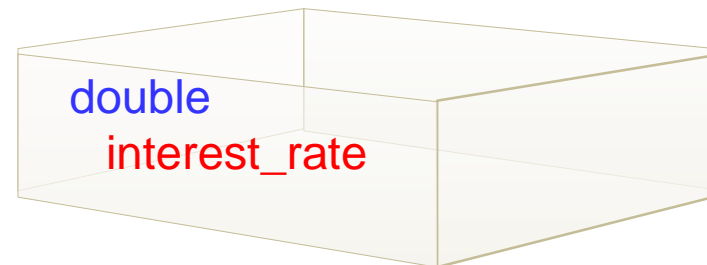
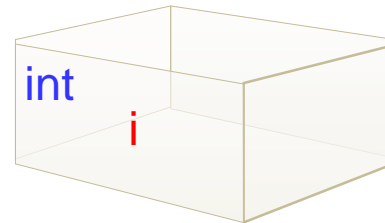
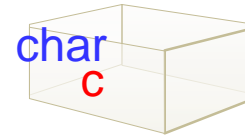
변수 이름

int index;



# 변수 선언의 예

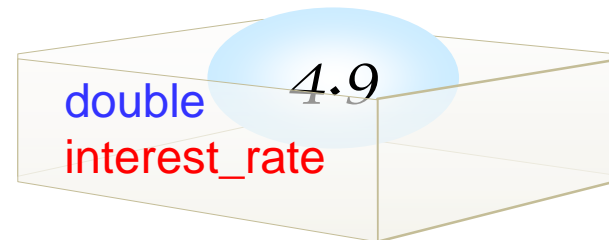
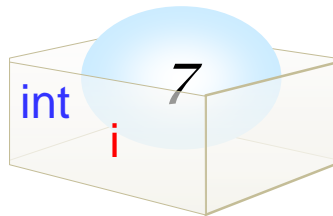
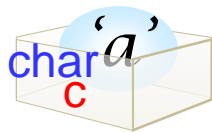
- 변수 선언의 예
  - `char c;`
  - `int i;`
  - `double interest_rate;`
  - `int height, width;`



# 변수에 값을 저장하는 방법

```
char c;           // 문자형 변수 c 선언
int i;            // 정수형 변수 i 선언
double interest_rate; // 실수형 변수 interest_rate 선언

c = 'a';          // 문자형 변수 c에 문자 'a'를 대입
i = 60;           // 정수형 변수 i에 60을 대입
interest_rate = 4.9; // 실수형 변수 interest_rate에 4.9를 대입
```

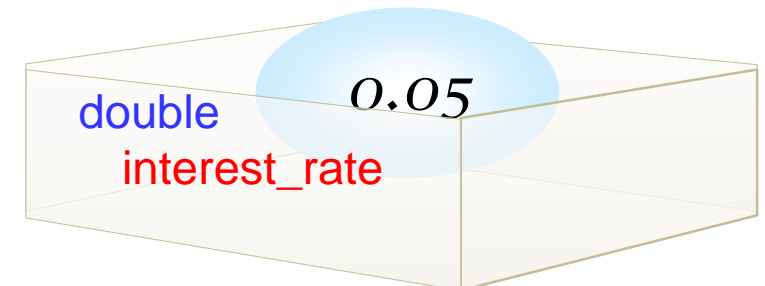
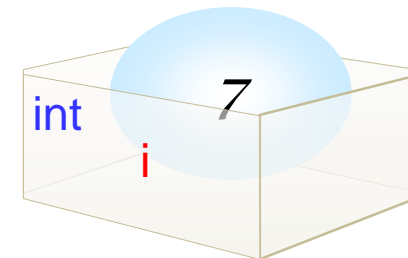
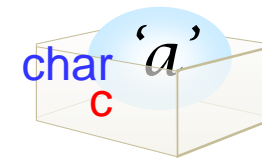


# 변수의 초기화

자료형    변수이름 = 초기값;

- 변수 초기화의 예

```
char c = 'a';  
int i = 7;  
double interest_rate = 0.05;
```





# 변수 선언 위치

```
int main(void)
{
    int i; // ○
```

```
    printf("Hello World!\n");
```

```
    ...
    int sum; // x
    ...
}
```

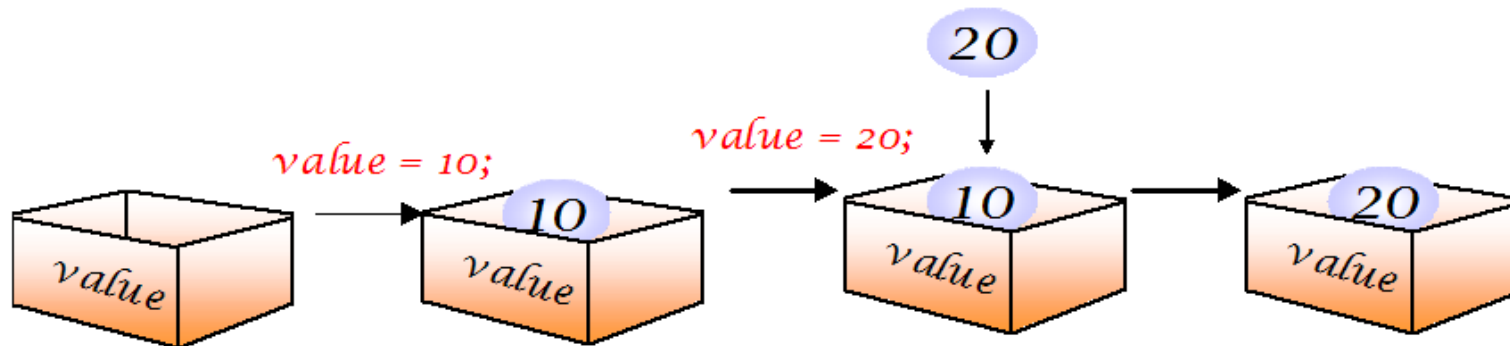
일반 문장이 시작된  
후에 변수를 선언하는  
것은 C언어에서는 곤  
란합니다.



# 변수의 사용

대입 연산자를 이용하여서 값을 저장한다.

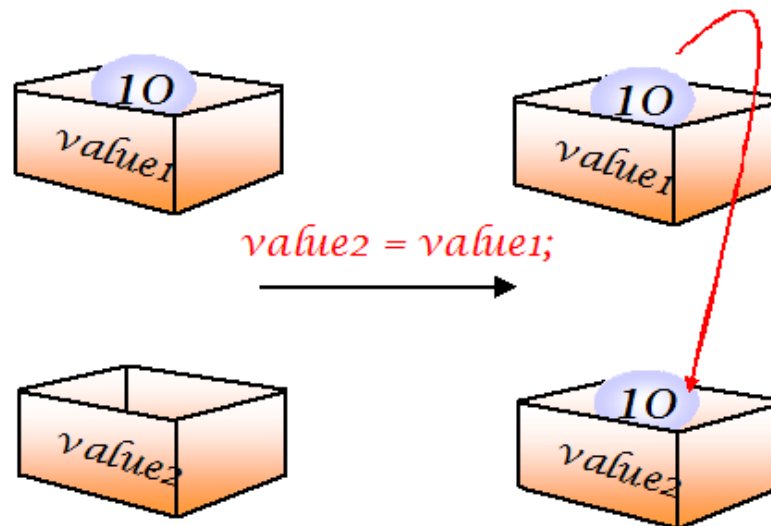
```
int value;  
value = 10;  
...  
value = 20;
```



# 변수의 사용

- 저장된 값은 변경이 가능하다.

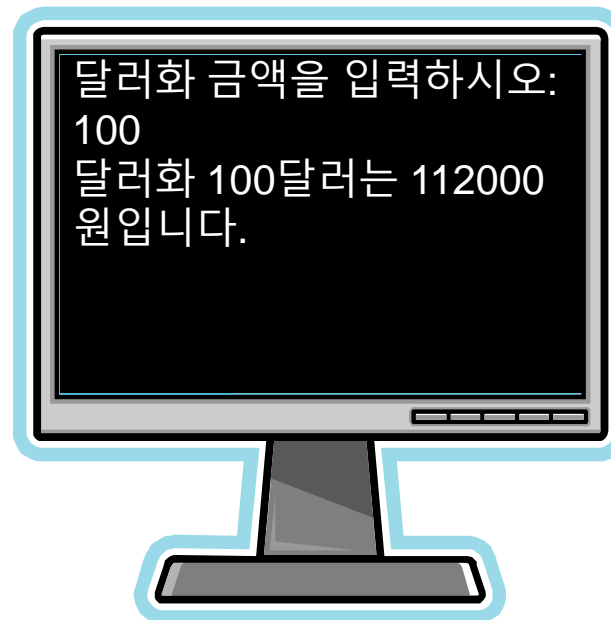
```
int value1 = 10;  
int value2;  
value2 = value1;
```



# 실습 예제: 변수 선언

---

- 1달러당 원화 1120원이라 가정
- 모든 변수의 자료형은 int 형
- 달러화 금액을 입력 받고 원화의 금액으로 변경해주어 기존 달러와 원화금액을 출력해주는 프로그램을 작성



# 예제:변수의 선언

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

```
int main(void)
```

```
{
```

```
    int usd;    // 달러화
```

```
    int krw;    // 원화
```

```
    printf("달러화 금액을 입력하시오: ");
```

```
    scanf("%d", &usd);
```

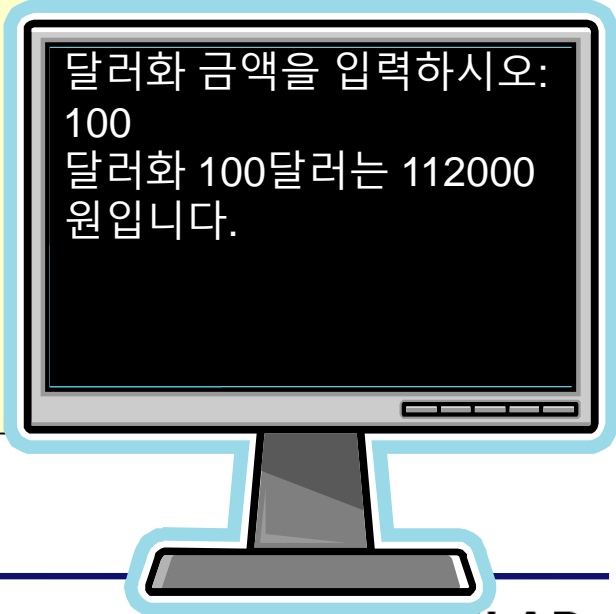
```
    krw = 1120 * usd;
```

```
    printf("달러화 %d달러는 %f원입니다.", usd, krw);
```

```
    return 0;
```

```
}
```

변수 선언



```
달러화 금액을 입력하시오:  
100  
달러화 100달러는 112000  
원입니다.
```

# 정수형

- short, int, long

8 bit=1 byte



16비트(2바이트) ≤ 32비트(4바이트) ≤ 32비트(4바이트)

- 가장 기본이 되는 것은 int
  - CPU에 따라서 크기가 달라진다.
  - 16비트, 32비트, 64비트

(Q) 왜 여러 개의 정수형이 필요한가?

(A) 용도에 따라 프로그래머가 선택하여 사용할 수 있게 하기 위하여

# 정수형 선언의 예

---

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

# 정수형의 범위

- int형

$$-2^{31}, \dots, -2, -1, 0, 1, 2, \dots, 2^{31} - 1$$

$$-2147483648 \leq n \leq +2147483647$$

약 -21억에서  
+21억

- short형

$$-2^{15}, \dots, -2, -1, 0, 1, 2, \dots, 2^{15} - 1$$

$$-32768 \leq n \leq +32767$$

- long형

- 보통 int형과 같음



# 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.

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

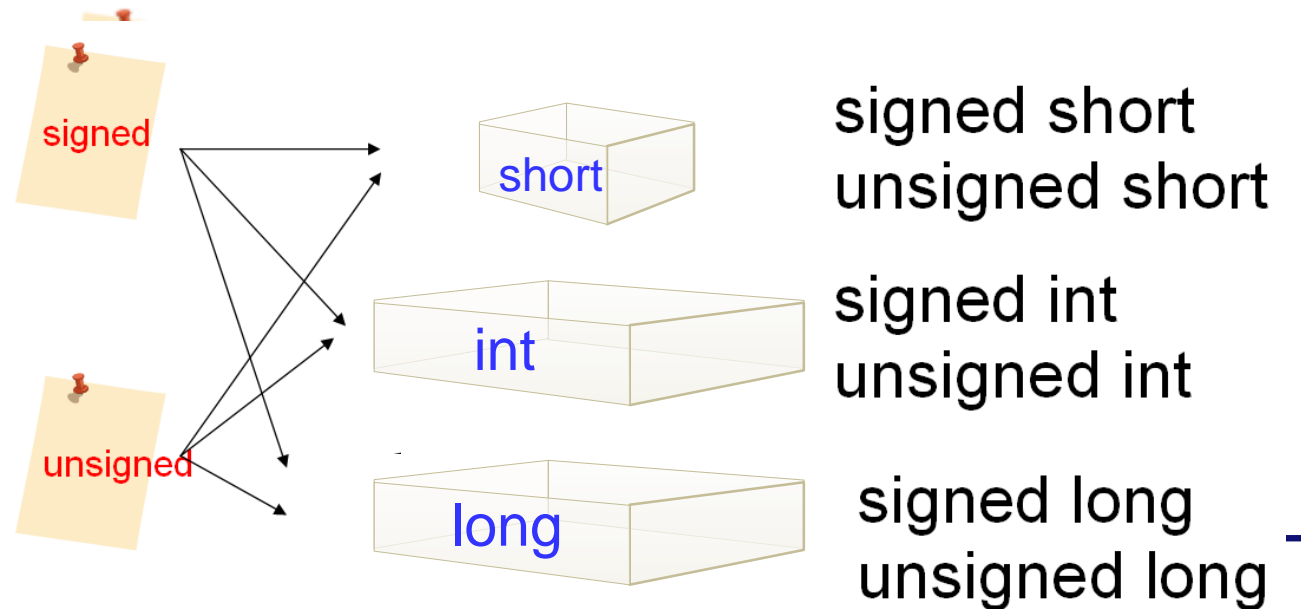
    printf("total_sale = %d \n", total_sale);

    return 0;
}
```



# signed, unsigned 수식자

- unsigned
  - 음수가 아닌 값만을 나타냄을 의미
  - unsigned int
$$0, 1, 2, \dots, 2^{32} - 1$$
$$(0 \sim +4294967295)$$
- signed
  - 부호를 가지는 값을 나타냄을 의미
  - 흔히 생략



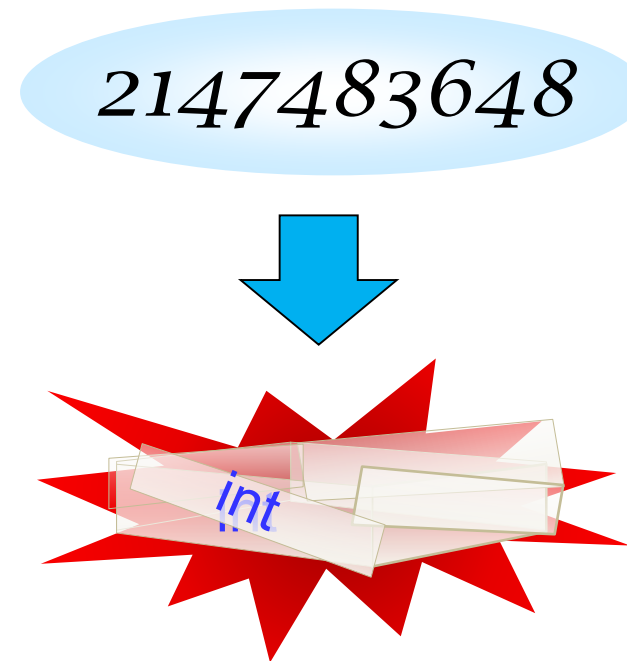
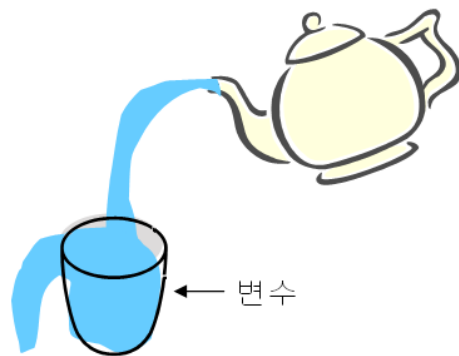
# unsigned 수식자

---

- *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



overflow

# 오버플로우

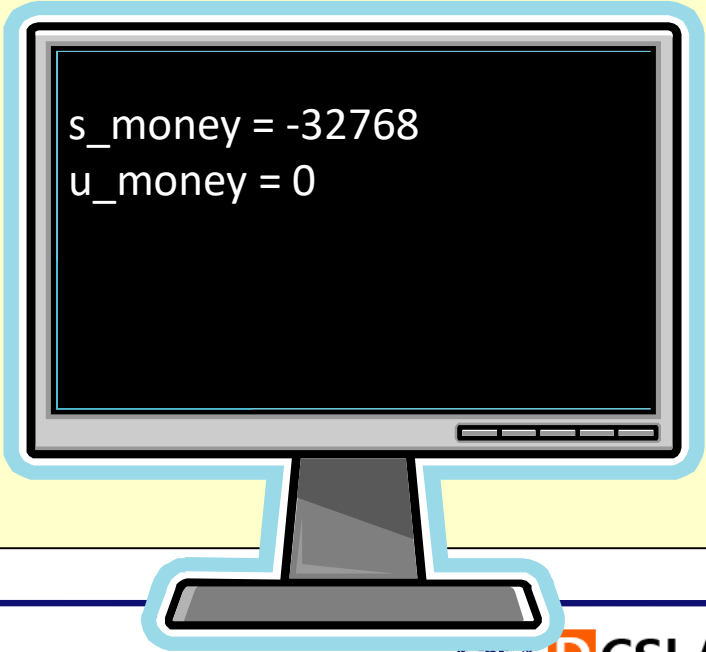
```
#include <stdio.h>
#include <limits.h>

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;
}
```

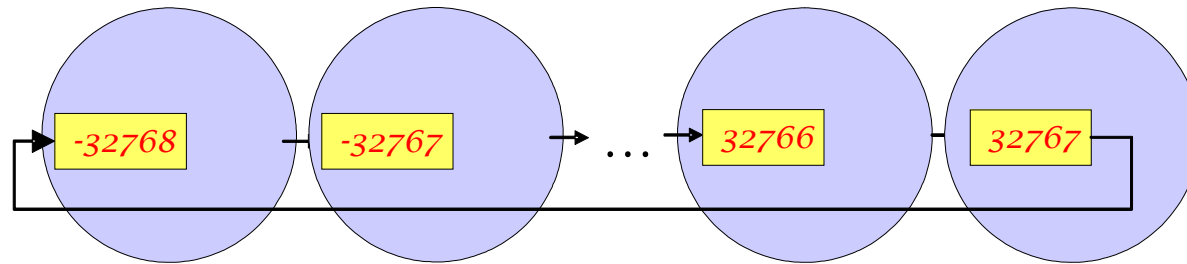
오버플로우 발생!!



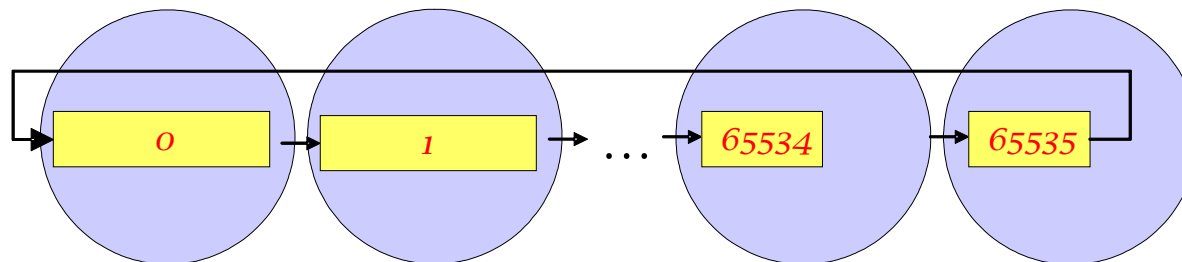
s\_money = -32768  
u\_money = 0

# 오버플로우

- 규칙성이 있다.
  - 수도 계량기나 자동차의 주행거리계와 비슷하게 동작



short의 경우



unsigned short 의 경우

# 오버플로우 언더플로우 예시

```
#include <stdio.h>

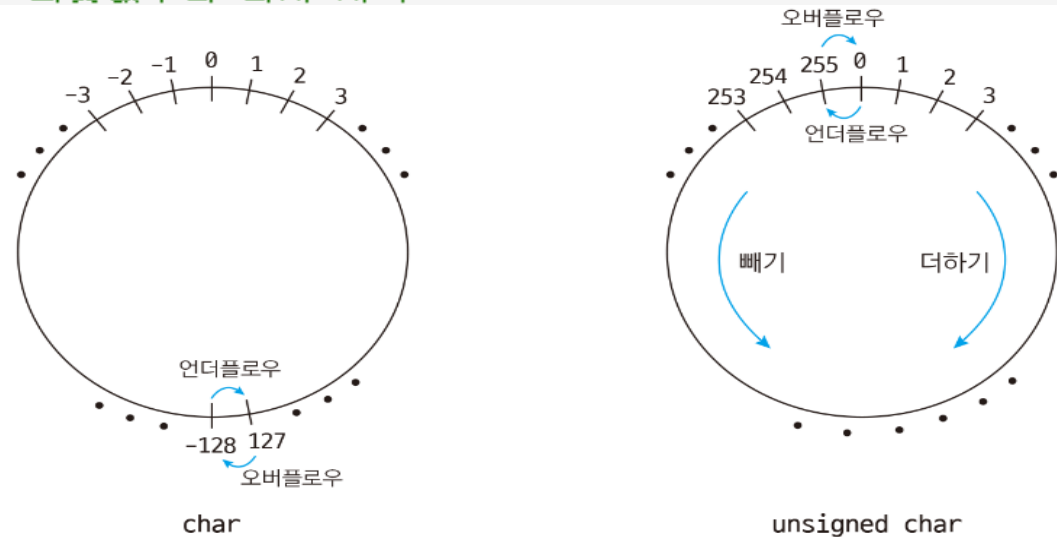
int main()
{
    char num1 = 128;           // char에 저장할 수 있는 최댓값 127보다 큰 수를 할당
                                // 오버플로우 발생

    unsigned char num2 = 256;  // unsigned char에 저장할 수 있는 최댓값 255보다 큰 수를 할당
                                // 오버플로우 발생

    printf("%d %u\n", num1, num2); // -128 0: 저장할 수 있는 범위를 넘어서므로
                                    // 최솟값부터 다시 시작

    return 0;
}
```

출력 값: -128 0



# 정수 상수

- 숫자를 적으면 기본적으로 int형 이 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

- 10진법 이외의 진법으로도 표기 가능

```
int x = 10;  
int y = 012;      // 8진수  
int z = 0xA;      // 16진수
```

$$012_8 = 1 \times 8^1 + 2 \times 8^0 = 10$$

$$0xA_{16} = 10 \times 16^0 = 10$$

자리수 증가

10진수	8진수	16진수
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	10	8
9	11	9
10	12	a
11	13	b
12	14	c
13	15	d
14	16	e
15	17	f
16	20	10
17	21	11

# 예제

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

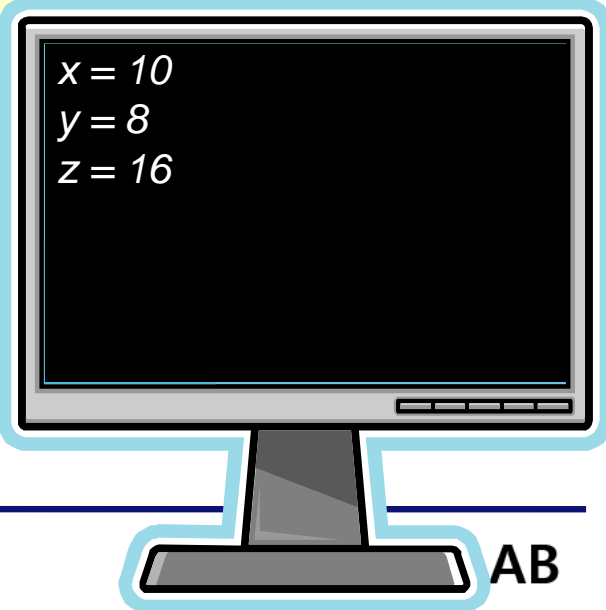
int main(void)
{
    int x = 10; // 10은 10진수이고 int형이고 값은 십진수로 10이다.
    int y = 010; // 010은 8진수이고 int형이고 값은 십진수로 8이다.
    int z = 0x10; // 010은 16진수이고 int형이고 값은 십진수로 16이다.

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

    return 0;
}
```

011=>9

0x21=>33



```
x = 10
y = 8
z = 16
```

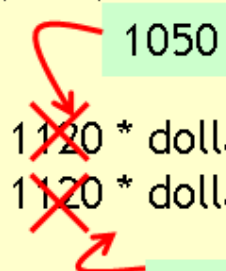
# 기호 상수

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

# 기호 상수의 장점

```
#include <stdio.h>

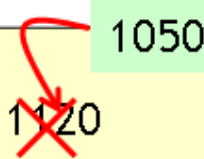
int main(void)
{
    ...
    won1 = 1120 * dollar1;
    won2 = 1120 * dollar2;
    ...
}
```



리터럴 상수를 사용하는 경우:  
등장하는 모든 곳을  
수정하여야한다.

```
#include <stdio.h>
#define EXCHANGE_RATE 1120

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

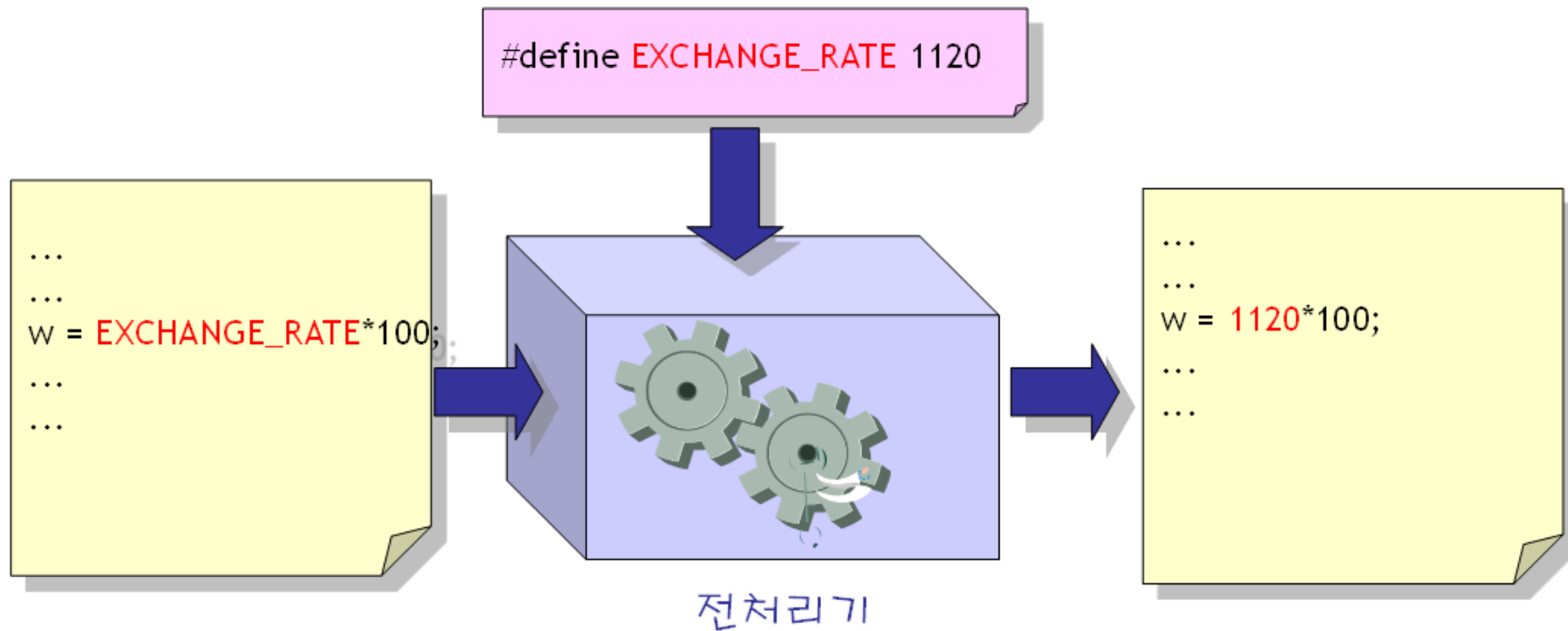


기호 상수를 사용하는 경우:  
기호 상수가 정의된 곳만 수정  
하면 된다.

# 기호 상수를 만드는 방법 #1

EXCHANGE\_RATE이라는 기호를 1120으로 정의

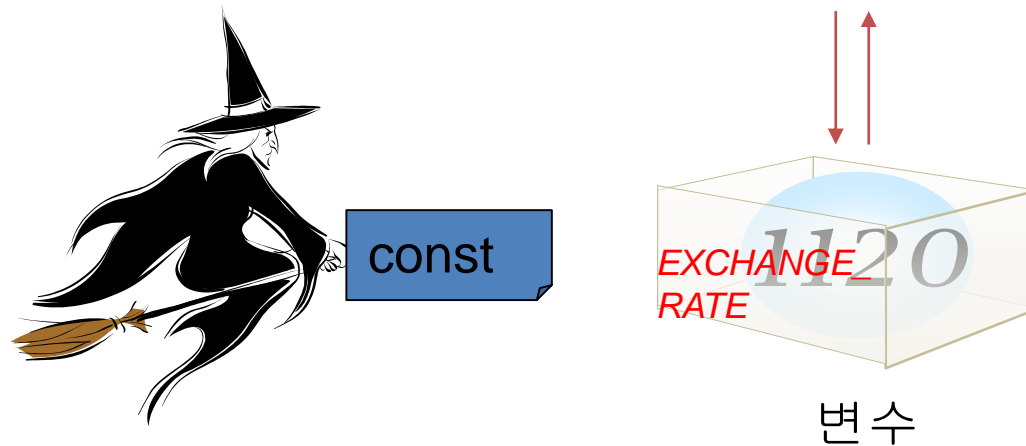
#define EXCHANGE\_RATE 1120



# 기호 상수를 만드는 방법 #2

변수가 값을 변경할 수 없게 한다.

const int EXCHANGE\_RATE = 1120;



# 예제 기호 상수

---

- 기호 상수를 사용하여 연봉 및 세금 계산
- 세금은 20퍼센트로 가정함
- Tax rate는 define을 사용하여 기호상수로 선언
- MONTHS(12개월)는 값을 변경하지 못하는 const를 사용하여 기호상수로 선언
- 월급을 입력 받고 연봉 출력
- 연봉에 해당하는 세금 출력



# 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);
```

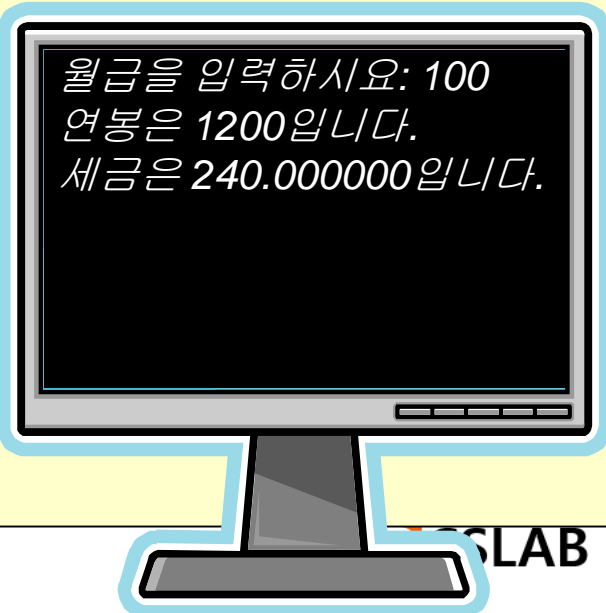
```
    y_salary = MONTHS * m_salary;      // 순수입 계산
```

```
    printf("연봉은 %d입니다.", y_salary);
```

```
    printf("세금은 %f입니다.", y_salary*TAX_RATE);
```

```
    return 0;
```

```
}
```

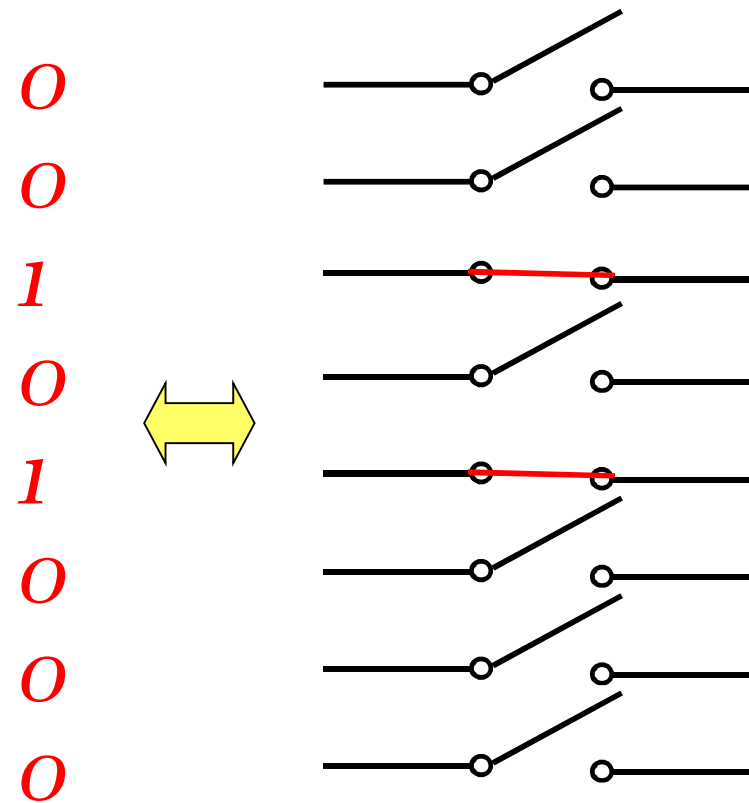


월급을 입력하시요: 100  
연봉은 1200입니다.  
세금은 240.000000입니다.



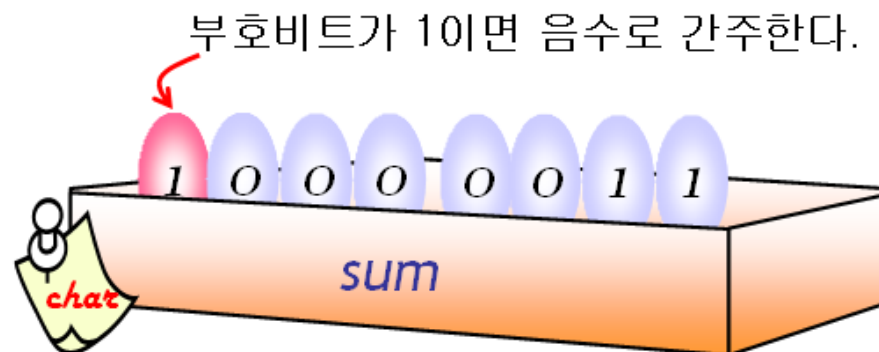
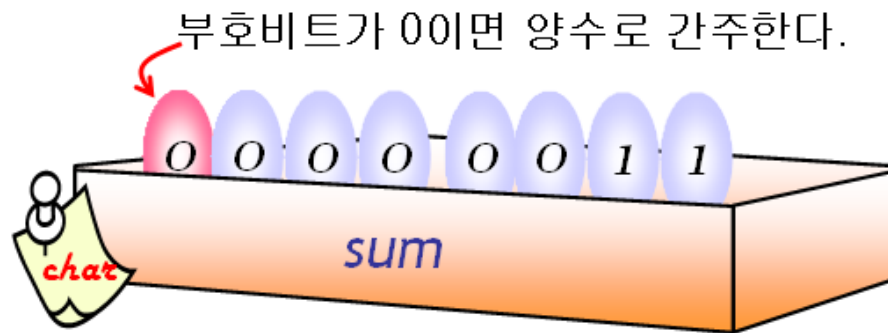
# 정수 표현 방법

컴퓨터에서 정수는 이진수 형태로 표현되고 이진수는 전자 스위치로 표현된다.



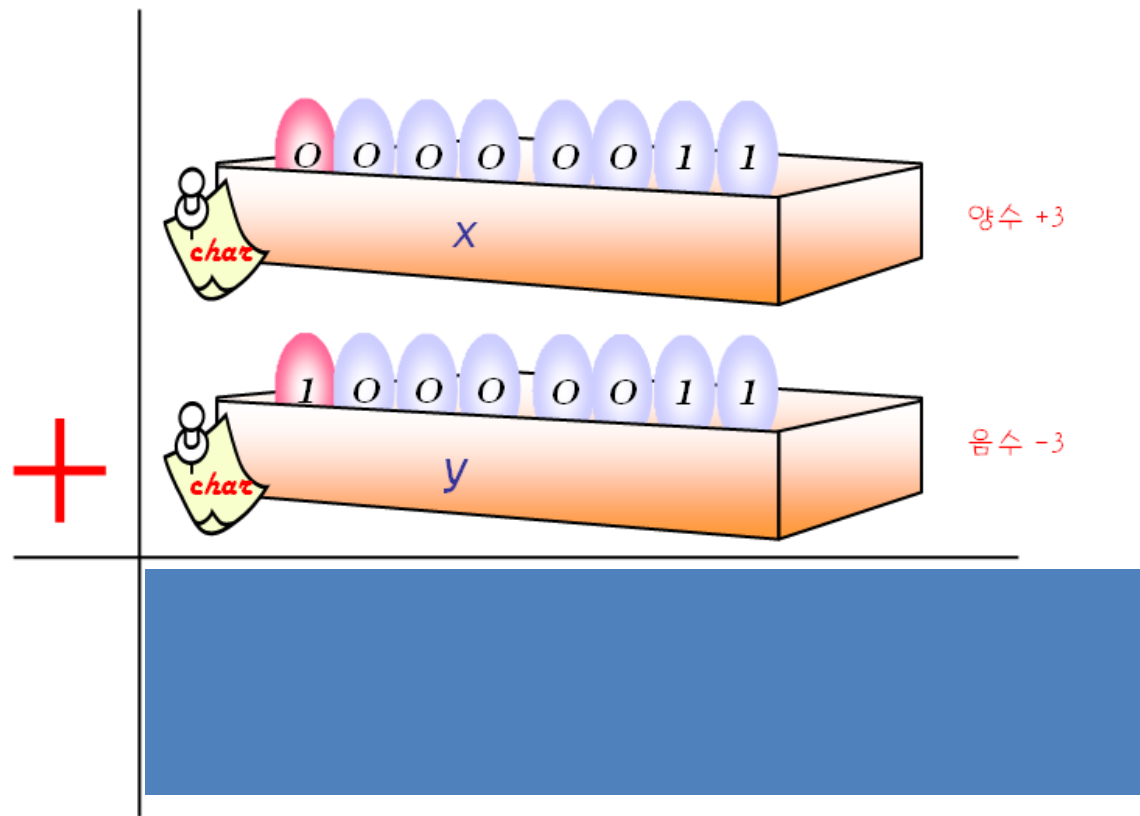
# 정수 표현 방법

- 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)  $+3 + (-3)$



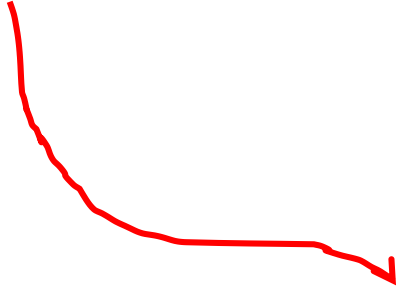
이 방법으로 표현  
된 이진수를  
평범하게 더하면  
결과가  
부정확합니다.



# 컴퓨터는 덧셈만 할 수 있다

---

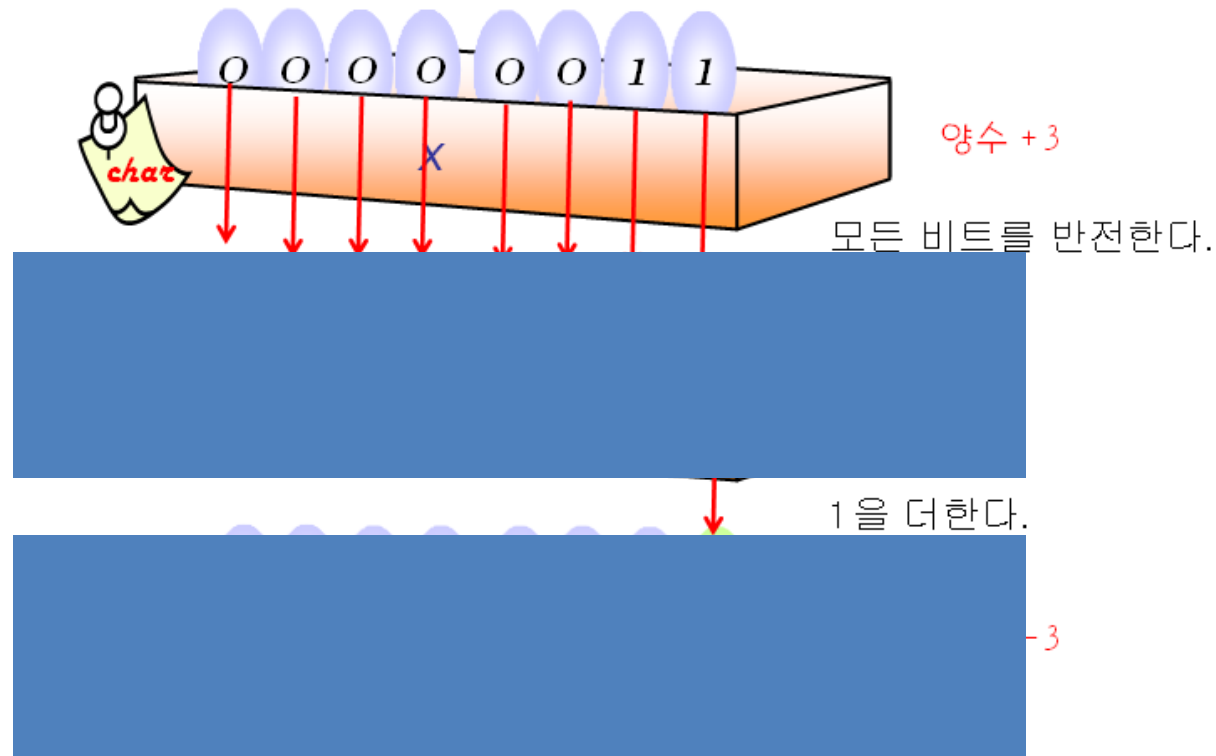
- 컴퓨터는 회로의 크기를 줄이기 위하여 덧셈회로만을 가지고 있다.
- 뺄셈은 다음과 같이 덧셈으로 변환한다.


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

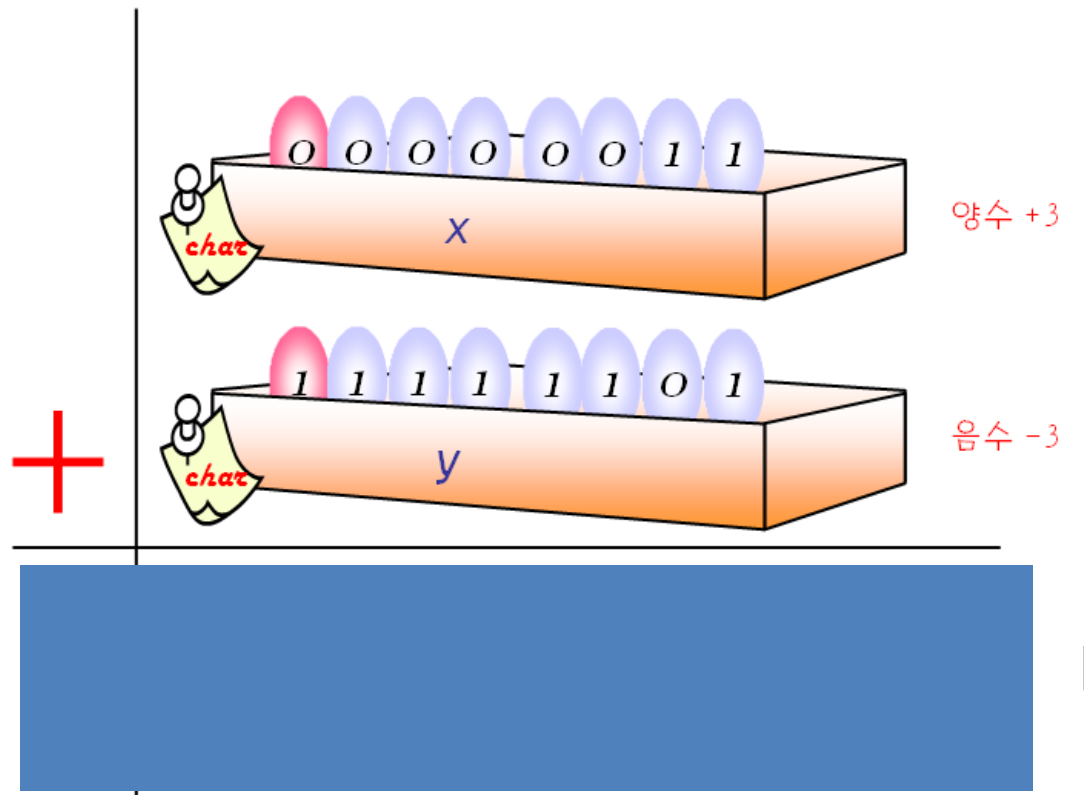
# 음수를 표현하는 두번째 방법

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

2의 보수(--補數, [영어](#): two's complement)란 어떤 수를 커다란 2의 제곱수에서 빼서 얻은 [이진수](#)이다.



# 2의 보수로 양수와 음수를 더하면



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



주어진 이진수보다 한 자리 높고 가장 높은 자리가 1이며 나머지가 0인 수에서 주어진 수를 빼서 얻은 수가 2의 보수이다.

# 예제

```
/* 2의 보수 프로그램*/
```

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

```
int main(void)
```

```
{
```

```
    int x = 3;
```

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

음수가 2의 보수로  
표현되는지를 알아보자,

```
    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;
```

```
}
```



x = 00000003

1 2 3 4 5 6 7 8 9 A B C D E F

y = FFFFFFFD

반전 후 => FFFFFFFC

x+y = 00000000

+1 => FFFFFFFD



# 부동소수점형

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

실수의 정밀도를 나타낸다.

실수의 표현범위를 나타낸다.

$$1.49598 \times 10^8$$

가수부분      지수부분

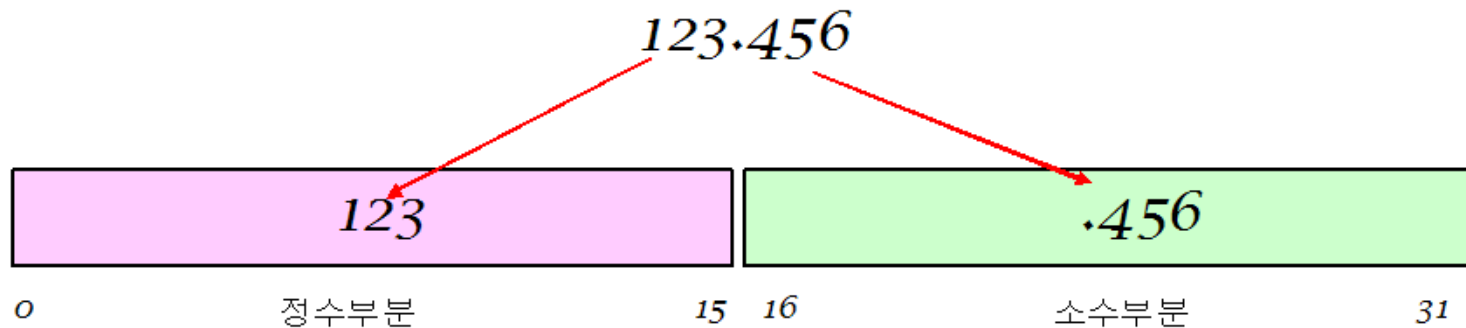
실수	과학적 표기법	지수 표기법
123.45	$1.2345 \times 10^2$	1.2345e2
12345.0	$1.2345 \times 10^5$	1.2345e5
0.000023	$2.3 \times 10^{-5}$	2.3e-5
2,000,000,000	$2.0 \times 10^9$	2.0e9



# 실수를 표현하는 방법

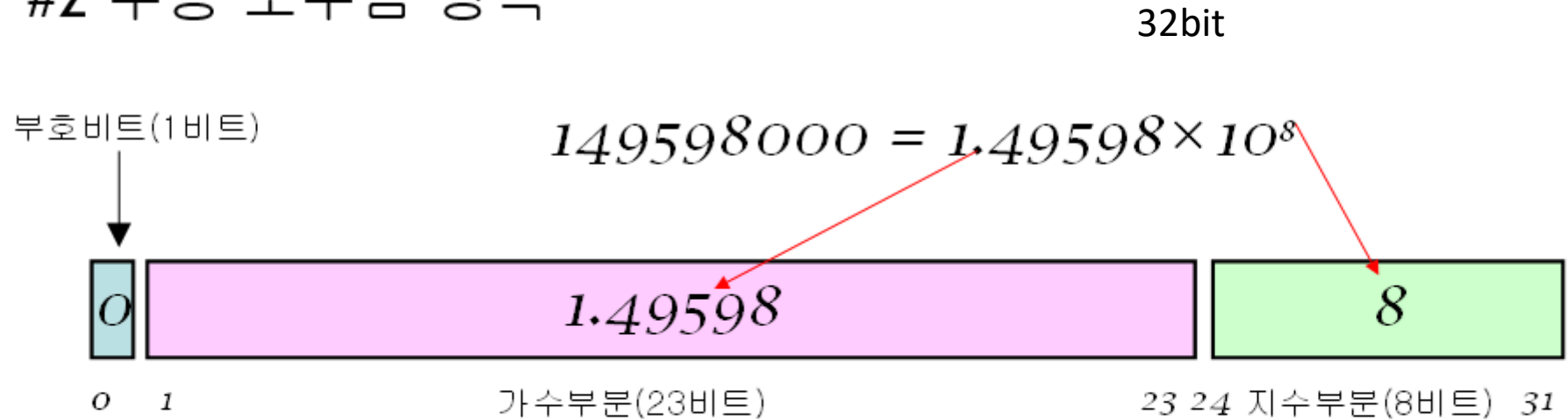
## #1 고정 소수점 방식

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  $10^{+38}$

2진수로 나타내는 지수범위 :  $2^{\text{exponent}}$

Exponent:  $2^8=128$

$2^{128} \approx 10^{38}$

# 부동 소수점 형



자료형	명칭	크기	범위
float	단일정밀도(single-precision) 부동소수점	32비트	$\pm 1.17549 \times 10^{-38} \sim \pm 3.40282 \times 10^{+38}$
double	두배정밀도(double-precision) 부동소수점	64비트	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{+308}$

# 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("long double의 크기=%d\n", sizeof(long double));

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



```
float의 크기=4
double의 크기=8
long double의 크기=8
x = 1.23456788063049320000000000
y = 1.23456789012345670000000000
```

# 부동 소수점 상수

---

- 일반적인 실수 표기법
  - 3.141592 (double형)
  - 3.141592F (float형)
- 지수표기법
  - $1.23456e4 = 12345.6$
  - $1.23456e-3 = 0.00123456$
- 유효한 표기법의 예
  - 1.23456
  - 2. // 소수점만 붙여도 된다.
  - .28 // 정수부가 없어도 된다.
  - 0e0
  - 2e+10 // +나 -기호를 지수부에 붙일 수 있다.
  - 9.26E3 //
  - 9.26e3 //

# 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;
```

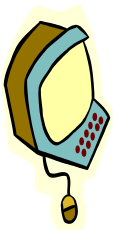
```
    printf("x = %e\n",x);
```

```
    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;
```

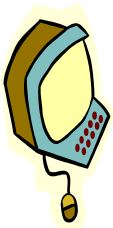
```
    x = (1.0e20 + 5.0)-1.0e20;
```

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

```
    return 0;
```

```
}
```

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



0.000000



# 중간 점검

1. 부동 소수점형에 속하는 자료형을 모두 열거하라.
2. float형 대신에 double형을 사용하는 이유는 무엇인가?
3. 부동 소수점형에서 오차가 발생하는 근본적인 이유는 무엇인가?
4. 12.345처럼 소수점이 있는 실수를 int형의 변수에 넣을 경우, 어떤 일이 발생하는가?
5. 수식  $(1.0/3.0)$ 을 float형 변수와 double형 변수에 각각 저장한 후에 출력하여 보자.  $(1.0/3.0)$ 은 0.333333.... 값을 출력하여야 한다. 소수점 몇 자리까지 정확하게 출력되는가?

0.33333334326744080000  
0.33333333333333331000 //소수점 20자리



# 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)
- 8비트를 사용하여 영어 알파벳 표현
- (예) !는 33, 'A'는 65, 'B'는 66, 'a'는 97, 'b'는 98

```
!"#$%&'()*+,-./0123456789:;<=>?  
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_  
`abcdefghijklmnopqrstuvwxyz{|}~
```

## 아스키 코드표 (일부)

Dec	Hex	문자
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;

Dec	Hex	문자
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

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

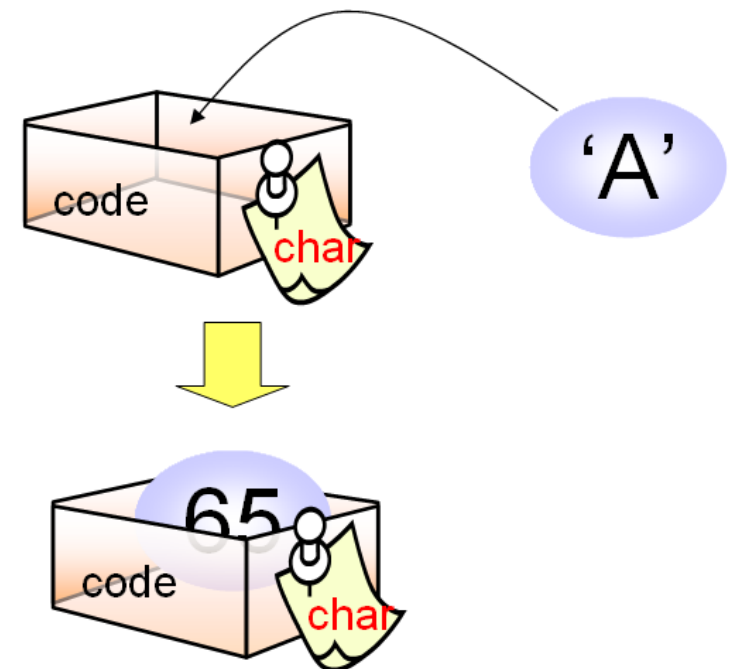
- char형의 변수가 문자 저장

```
char c;  
char answer;  
char code;
```



- char형의 변수에 문자를 저장하려면 아스키 코드 값을 대입

```
code = 65;           // 'A' 저장  
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("문자 상수 초기화 = %c\n", code1);  
    printf("아스키 코드 초기화 = %c\n", code2);  
}
```



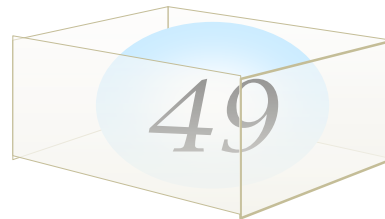
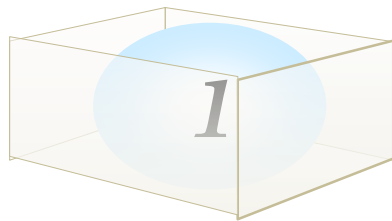
문자 상수 초기화 = A  
아스키 코드 초기화 = A

# 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



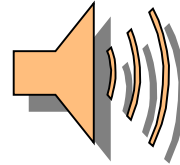


# 제어 문자를 나타내는 방법

---

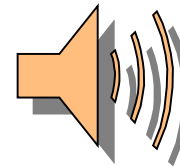
- 아스키 코드를 직접 사용

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



- 이스케이프 시퀀스 사용

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



# 이스케이프 시퀀스

제어 문자 이름	제어 문자 표기	값	의미
널문자	\0	0	문자열의 끝을 표시
경고(bell)	\a	7	"삐"하는 경고 벨소리 발생
백스페이스(backspace)	\b	8	커서를 현재의 위치에서 한 글자 뒤로 옮긴다.
수평탭(horizontal tab)	\t	9	커서의 위치를 현재 라인에서 설정된 다음 탭 위치로 옮긴다.
줄바꿈(newline)	\n	10	커서를 다음 라인의 시작 위치로 옮긴다.
수직탭(vertical tab)	\v	11	설정되어 있는 다음 수직 탭 위치로 커서를 이동
폼피드(form feed)	\f	12	주로 프린터에서 강제로 다음 페이지로 넘길 때 사용된다.
캐리지 리턴(carriage return)	\r	13	커서를 현재 라인의 시작 위치로 옮긴다.
큰따옴표	\“	34	원래의 큰따옴표 자체
작은따옴표	\‘	39	원래의 작은따옴표 자체
역슬래시(back slash)	\\	92	원래의 역슬래시 자체

# 예제

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

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

    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;
}
```

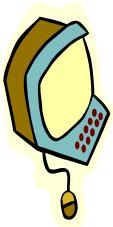


아이디와 패스워드를 4개의 숫자로 입력하세요:  
id: 1234  
pass: 5678  
입력된 아이디는 "1234"이고 패스워드는 "5678"입니다.

# 정수형으로서의 char형

- 8비트의 정수를 저장하는데 char 형을 사용할 수 있다..

```
char code = 65;  
printf("%d %d %d", code, code+1, code+2); // 65 66 67이 출력된다.  
printf("%c %c %c", code, code+1, code+2); // A B C가 출력된다.
```



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.

```
int main(void)
{
    printf("\a 화재가 발생하였습니다. \n");

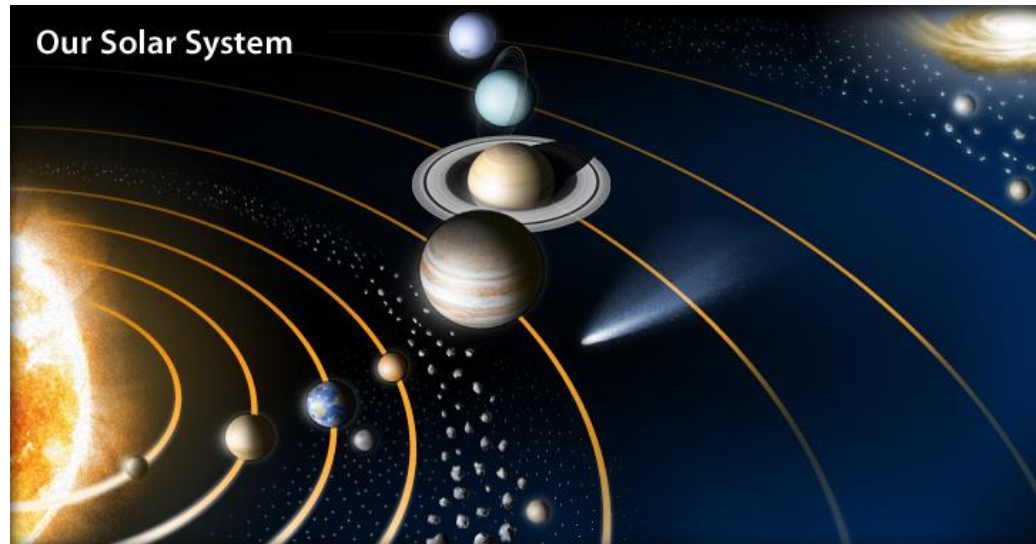
    return 0;
}
```



# 실습: 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.



```
#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 %f분\n", time);    // 시간을 출력한다.

    return 0;
}
```



```
빛의 속도는 300000.000000km/s
태양과 지구와의 거리 149600000.000000km
도달 시간은 8.311111분
```

# 도전문제

---

- 위의 프로그램의 출력은 8.31111...분로 나온다. 이것을 분과 초로 나누어서 8분 18초와 같은 식으로 출력하도록 변경하라. 필요하다면 형변환을 사용하라. 추가적인 정수 변수를 사용하여도 좋다.

힌트

`#define hour 3600 // 1시간 3600초`

`#define min 60 // 1분 60초`

`% => 나누고 나머지를 나타냄`



# Q & A

---

