

쉽게 풀어쓴 C언어 Express[개정판] 천인국 저, 생능출판사 2012

4장. 변수와 자료형

성결대학교 컴퓨터공학부
임 상 순

강의 목표 및 내용

▶ 강의 목표

- 변수와 상수의 개념을 이해한다.
- C에서 사용가능한 변수의 종류를 알고 있다.
- 정수형 변수와 상수를 선언하고 사용할 수 있다.
- 부동 소수점형 변수와 상수를 선언하고 사용할 수 있다.
- 기호 상수를 사용할 수 있다.
- 오버플로우와 언더플로우를 이해한다.

▶ 내용

- 변수와 상수
- 자료형
- 변수의 이름 짓기
- 변수 선언과 사용
- 정수형, 부동 소수점형, 문자형

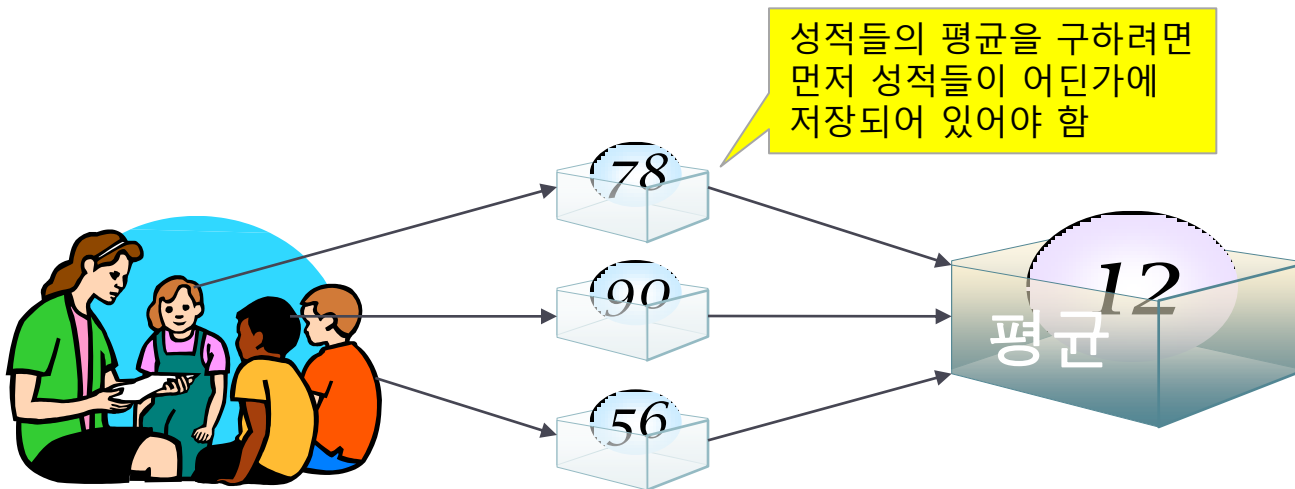
변수의 기본 개념[1/4]

Q) 변수(variable)란 무엇인가?

A) 프로그램에서 일시적으로 데이터를 저장하는 공간

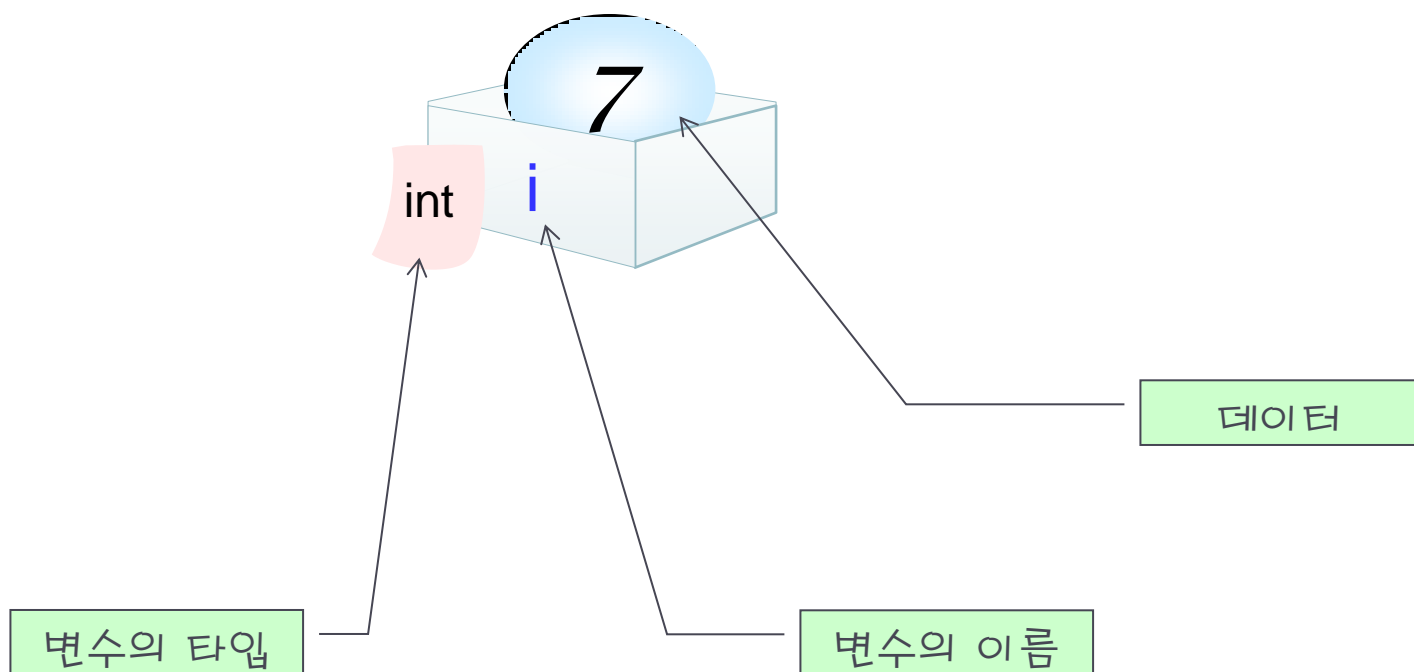
Q) 변수는 왜 필요한가?

A) 데이터가 입력되면 어딘가에 저장해야만 다음에 사용할 수 있다.



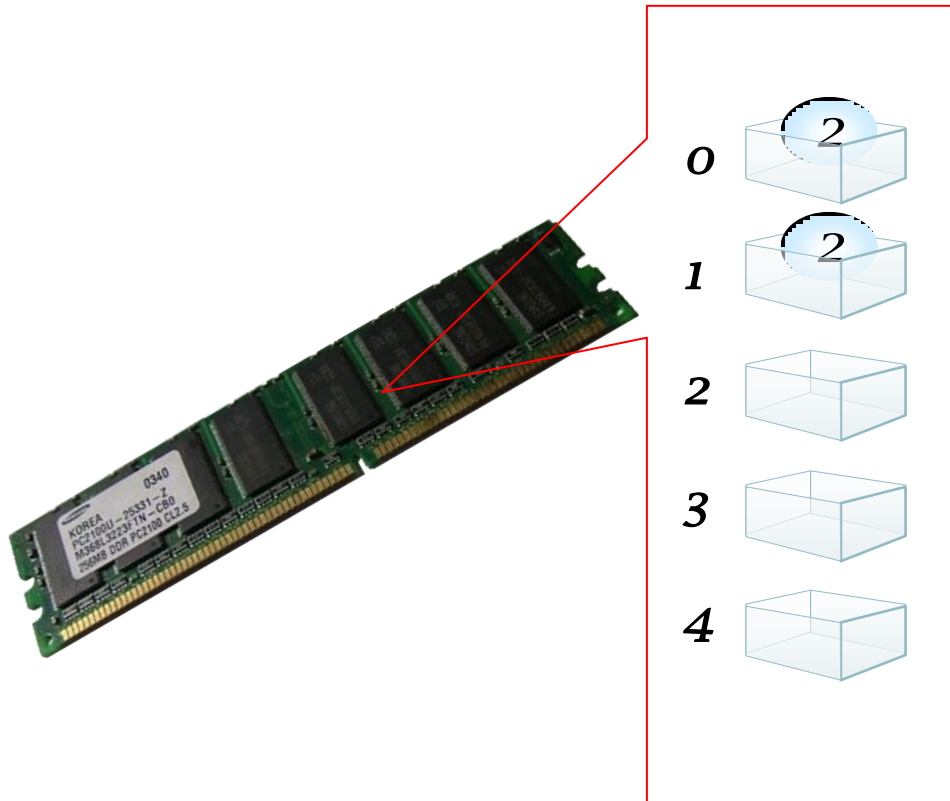
변수의 기본 개념[2/4]

- ▶ 변수는 물건을 저장하는 상자와 같음
 - 변수의 값(물건)은 언제든지 다른 값으로 변경 가능
 - 변수를 식별하기 위해 변수에 이름을 붙임



변수의 기본 개념[3/4]

- ▶ 변수는 메인 메모리에 저장됨

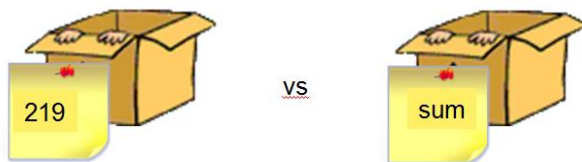


변수의 기본 개념 [4/4]

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

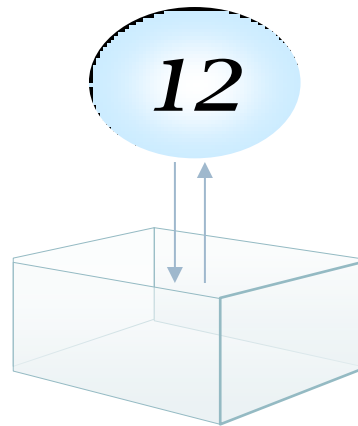
“100번지에 0을 대입하라”

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

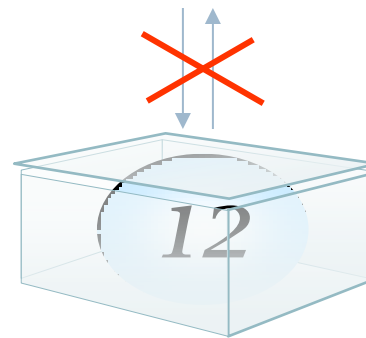


변수와 상수

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



변수

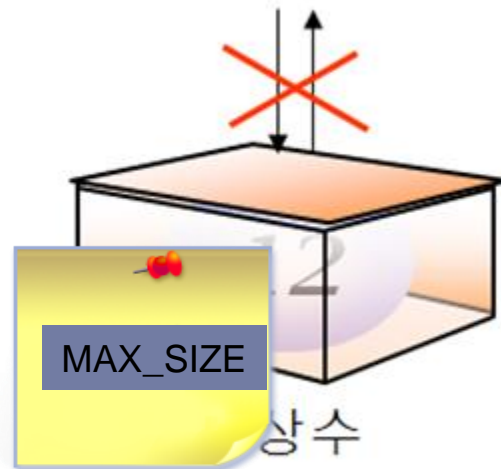


상수

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(circle_area.c)

```
/* 원의 면적을 계산하는 프로그램 */
```

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

```
int main(void)
```

```
{
```

```
float radius;
```

```
float area;
```

// 원의 반지름

// 원의 면적

```
printf("원의 면적을 입력하시요:");
```

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

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

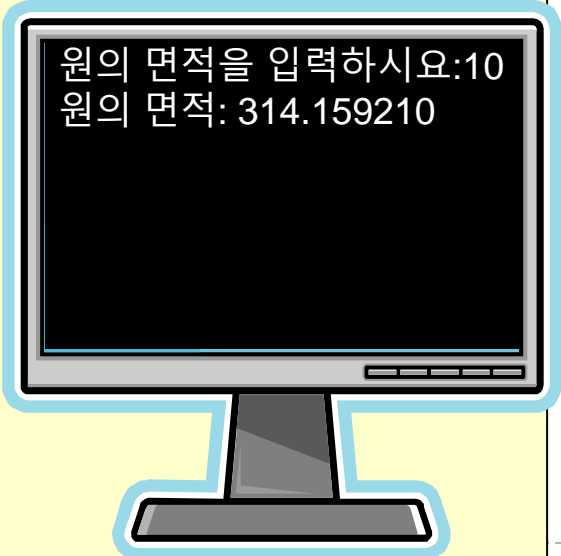
```
printf("원의 면적: %f\n", area);
```

```
return 0;
```

```
}
```

변수

상수



원의 면적을 입력하시요:10
원의 면적: 314.159210

자료형 [1/4]

▶ 자료형(data type)

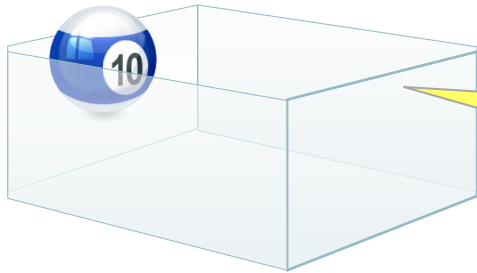
- 데이터의 종류에 따라 저장하는데 필요한 메모리 공간이 다름
 - ▶ 데이터 종류에 따라 변수의 종류를 다르게 하는 것이 효율적
- 데이터의 타입(종류)
 - ▶ (예) short, int, long: 정수형 데이터(100)
 - ▶ (예) double, float: 실수형 데이터(3.141592)
 - ▶ (예) char: 문자형 데이터('A', 'a', '한')



자료형 [2/4]

(Q) 다양한 자료형이 필요한 이유는?

(A) 상자에 물건을 저장하는 것과 같다.

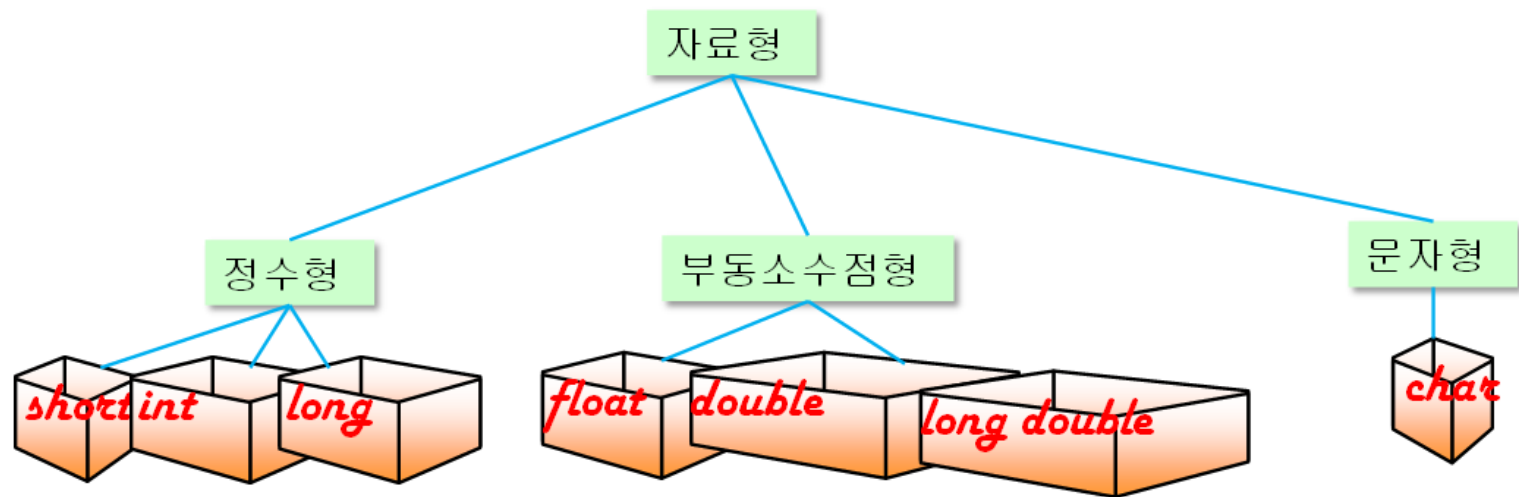


물건이 상자보다 너무 작으면 공간이 낭비될 것이다.

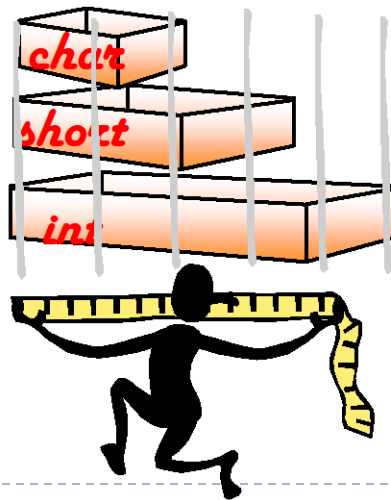
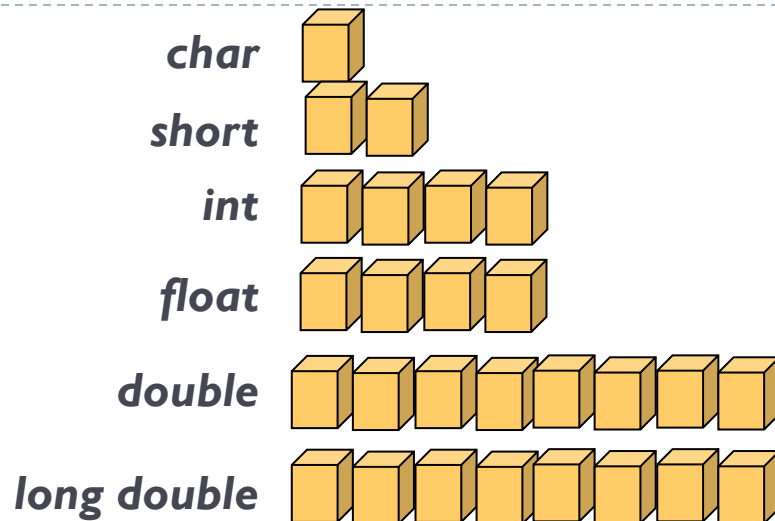


물건이 상자보다 크면 들어가지 않을 것이다.

자료형 [3/4]



자료형 [4/4]



sizeof 연산자는 변수나 데이터 타입의 크기를 바이트 단위로 반환합니다.



예제 : 자료형의 크기(sizeof.c)

```
#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));
```

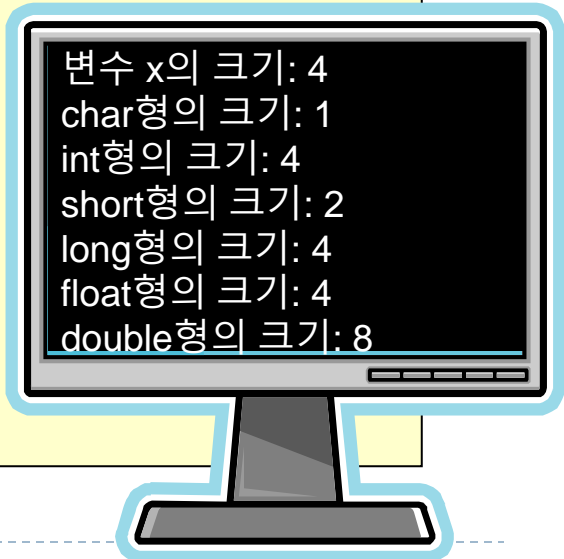
```
    printf("long형의 크기: %d\\n", sizeof(long));
```

```
    printf("float형의 크기: %d\\n", sizeof(float));
```

```
    printf("double형의 크기: %d\\n", sizeof(double));
```

```
    return 0;
```

```
}
```



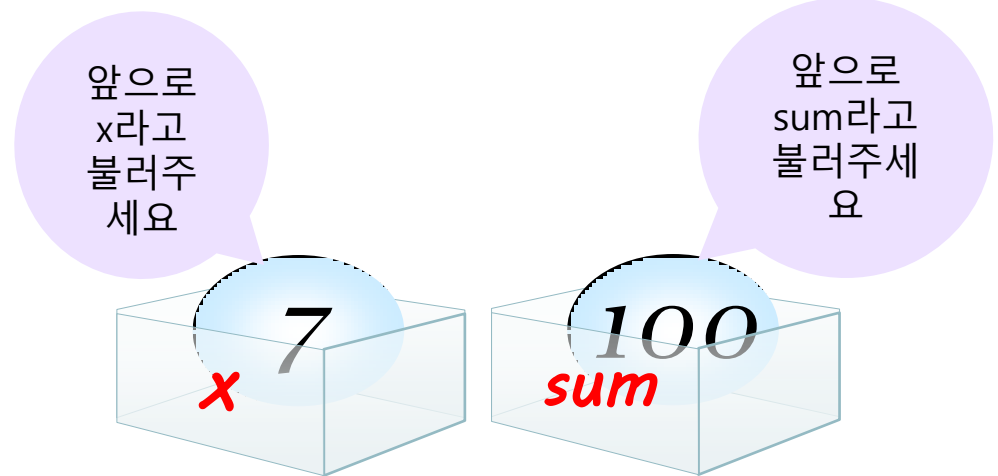
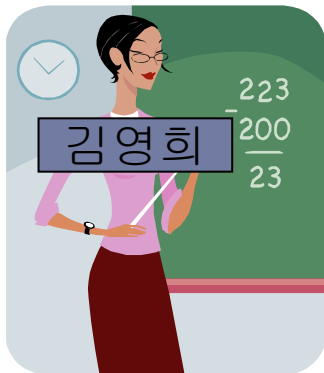
변수 x의 크기: 4
char형의 크기: 1
int형의 크기: 4
short형의 크기: 2
long형의 크기: 4
float형의 크기: 4
double형의 크기: 8

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 이름 짓기[1/4]

- ▶ 변수는 반드시 이름이 있어야 함
 - 식별자(identifier)
 - ▶ 식별할 수 있게 해주는 이름
 - ▶ 변수 이름
 - ▶ 함수 이름



변수의 이름 짓기[2/4]

▶ 식별자 생성 규칙

- 알파벳 문자와 숫자, 밑줄 문자 _로 구성
- 중간에 공백이 들어가면 안됨
- 첫 번째 문자는 반드시 알파벳 또는 밑줄 문자 _
- 대문자와 소문자를 구별
- C 언어의 키워드와 똑같은 이름은 허용되지 않음

(Q) 다음은 유효한 식별자인가?

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

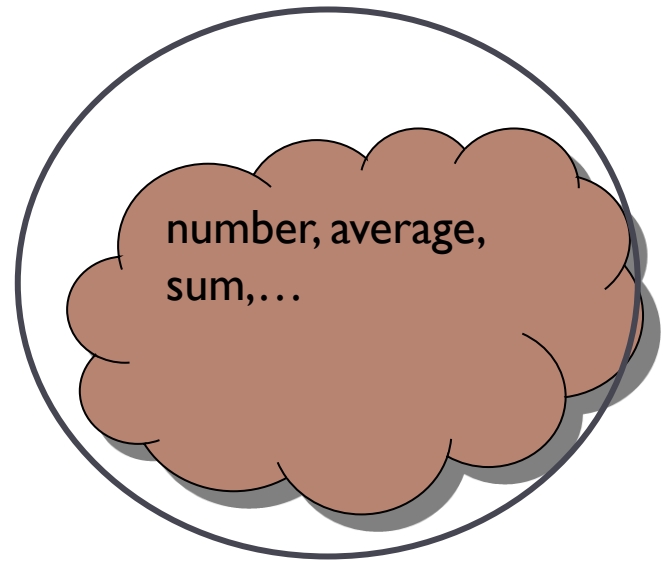
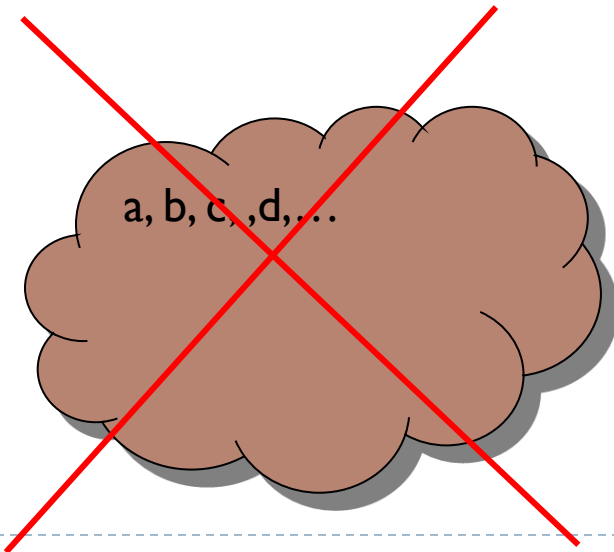
변수의 이름 짓기 [3/4]

- ▶ 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

변수의 이름 짓기[4/4]

- ▶ 변수의 이름을 짓는 것은 상당히 중요한 작업
 - 프로그램의 가독성에 중요한 역할
 - 일부 회사들은 코딩 규약에 변수 이름 생성 규칙을 명시함
- ▶ 변수의 역할을 가장 잘 설명하는 이름으로 작성
 - 밑줄 방식: `bank_account`
 - 단어의 첫 번째 글자를 대문자: `BankAccount`



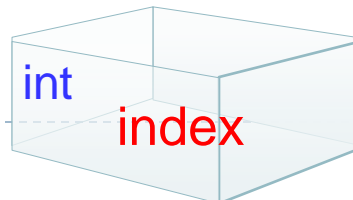
변수 선언[1/3]

- ▶ 변수는 사용하기 전에 반드시 미리 선언(declare) 해야 함
 - 컴파일러에게 어떤 변수를 사용하겠다고 미리 알리는 것
 - 변수 선언을 하게 되면 컴파일러는 변수를 저장하는 메모리 공간을 미리 확보
- ▶ 변수 선언 방법
 - (형식) 자료형 변수이름;

자료형

변수 이름

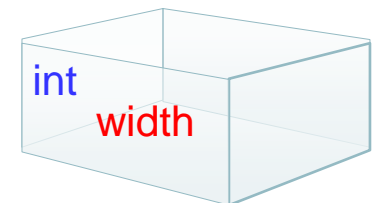
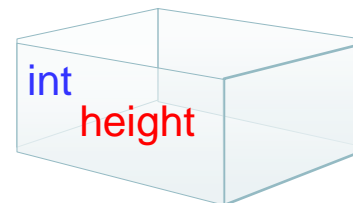
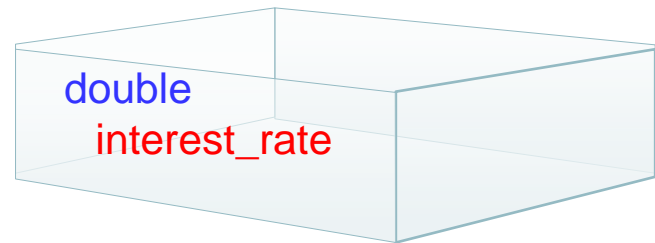
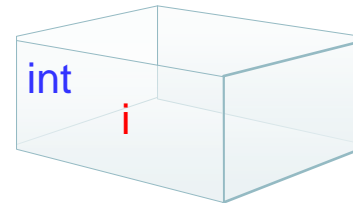
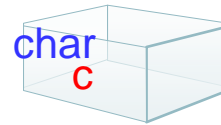
int index;



변수 선언 [2/3]

▶ 변수 선언의 예

- char c;
- int i;
- double interest_rate;
- int height, width;



변수 선언[3/3]

- ▶ 변수는 반드시 함수의 시작 부분에서 선언
 - 시작 부분이 아닌 곳에서 선언되면 컴파일 오류 발생(예전 버전)
 - 최신 버전인 C11에서는 어디에서든지 변수를 선언 가능

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

    printf("Hello World!\n");
    ...
    int sum;         // ✕
    ...
}
```

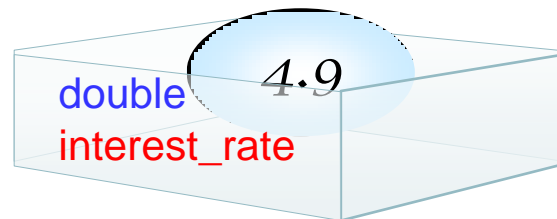
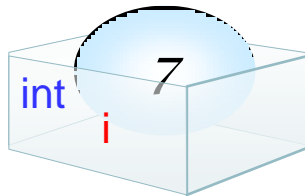
일반 문장이 시작된
후에 변수를 선언하
는 것은 C언어에서
곤란합니다.



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
```



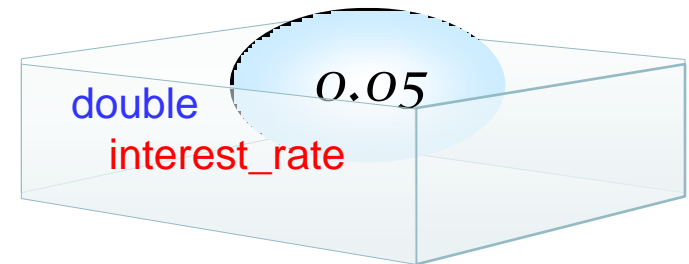
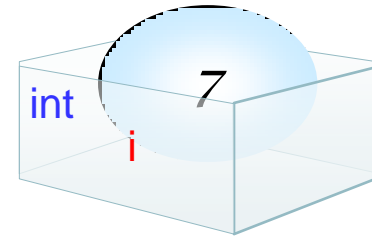
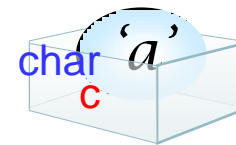
변수의 초기화

- ▶ 변수를 선언과 동시에 값을 넣는 방법
 - 변수 이름 뒤에 =을 붙이고 초기값을 적음

자료형 변수이름 = 초기값;

- ▶ 변수 초기화의 예

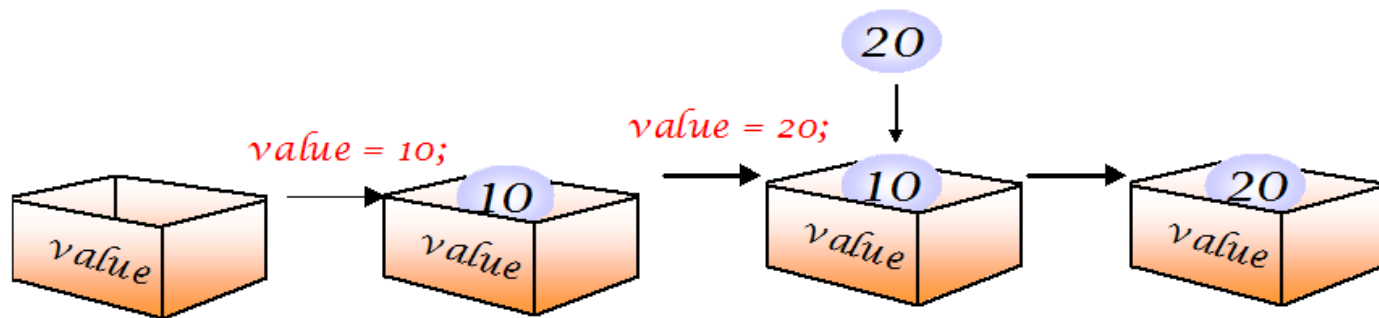
```
char c = 'a';  
int i = 7;  
double interest_rate = 0.05;
```



변수의 사용[1/2]

- ▶ 대입 연산자를 이용하여서 값을 저장
- ▶ 여러 번 변경 가능

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



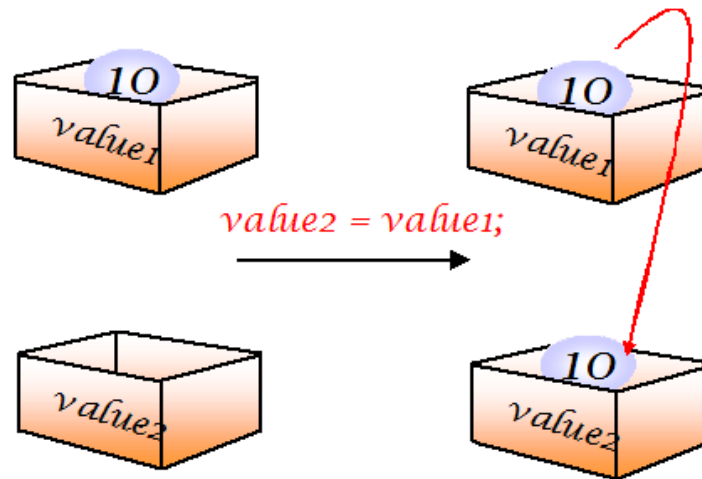
변수의 사용[2/2]

- ▶ 변수에는 다른 변수의 값도 대입 가능

```
int value1 = 10;
```

```
int value2;
```

```
value2 = value1;
```



예제: 변수의 선언(dollar2won.c)

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

변수 선언

```
int main(void)
```

```
{
```

```
    int usd;    // 달러화
```

```
    int krw;    // 원화
```

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

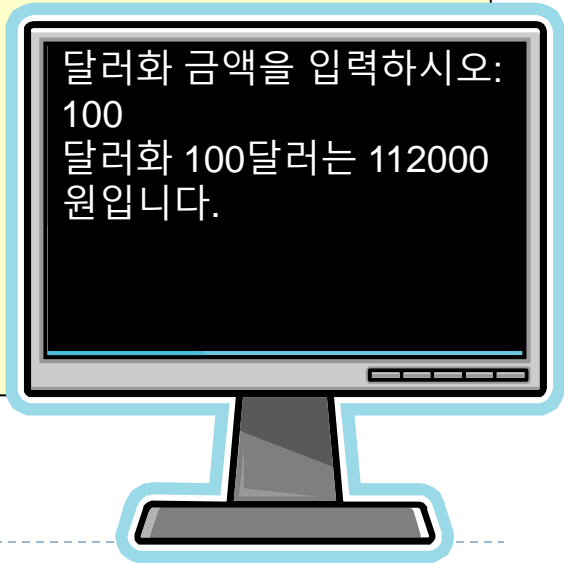
```
    scanf("%d", &usd);
```

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

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

```
    return 0;
```

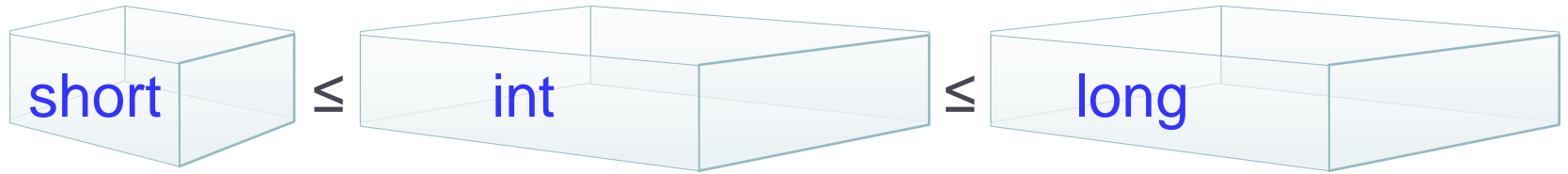
```
}
```



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

정수형

- ▶ short, int, long



16비트(2바이트) \leq 32비트(4바이트) \leq 32비트(4바이트)

- ▶ 가장 기본이 되는 것은 int
 - CPU에 따라서 크기가 달라진다.
 - 16비트, 32비트, 64비트
- ▶ 여러 개의 정수형이 필요한 이유
 - 용도에 따라 프로그래머가 선택하여 사용할 수 있게 하기 위해

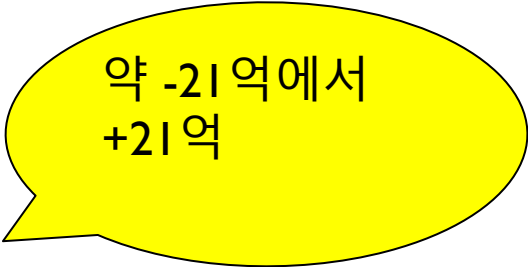
정수형 선언

- ▶ `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 : 정수 자료형 사용(integer.c)

```
/* 정수 자료형을 사용하는 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

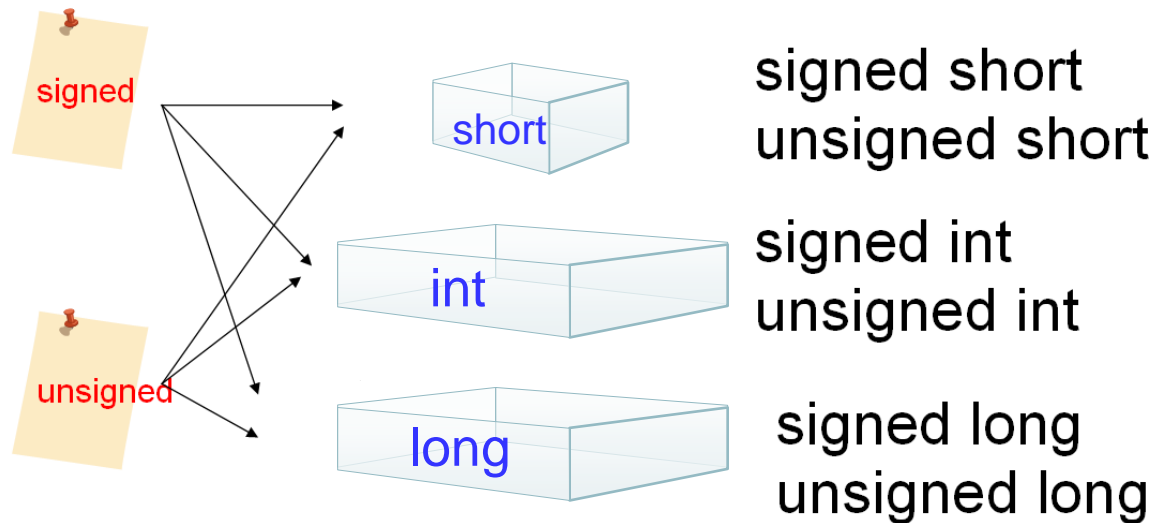
▶ unsigned

- 음수가 아닌 값만을 나타냄을 의미
- unsigned int

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

▶ signed

- 부호를 가지는 값을 나타냄을 의미
- 흔히 생략



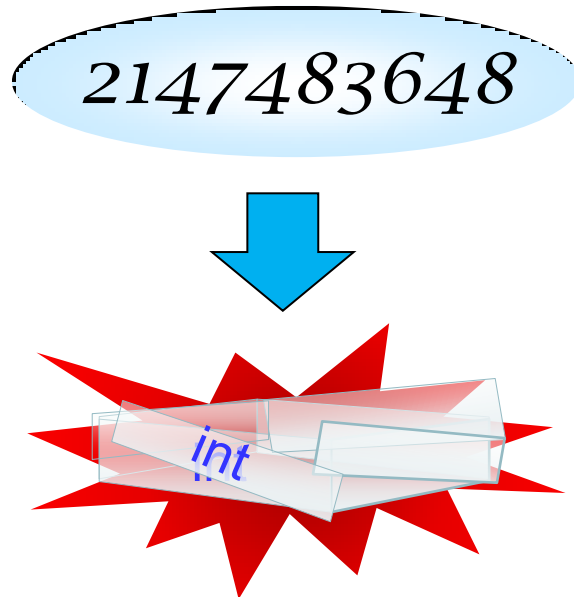
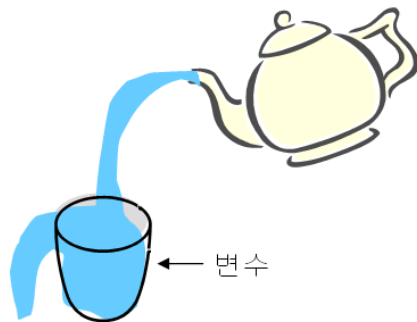
unsigned 수식자 사용 예

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

오버플로우[1/2]

▶ 오버플로우(overflow)

- 변수가 나타낼 수 있는 범위를 넘는 숫자를 저장하려고 할 때 발생
- int형 변수에 저장할 수 있는 2147483647을 넘어가는 값을 저장한다면?



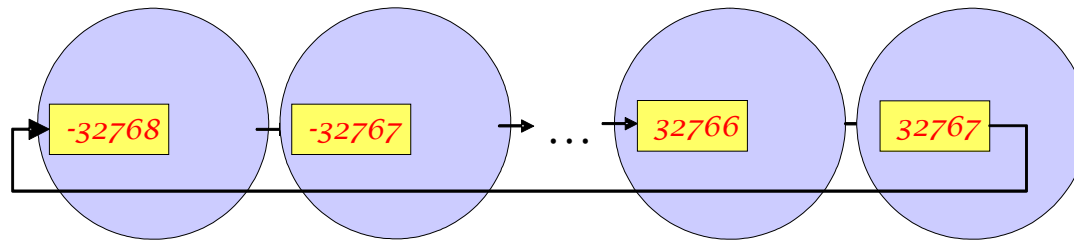
overflow

오버플로우[2/2]

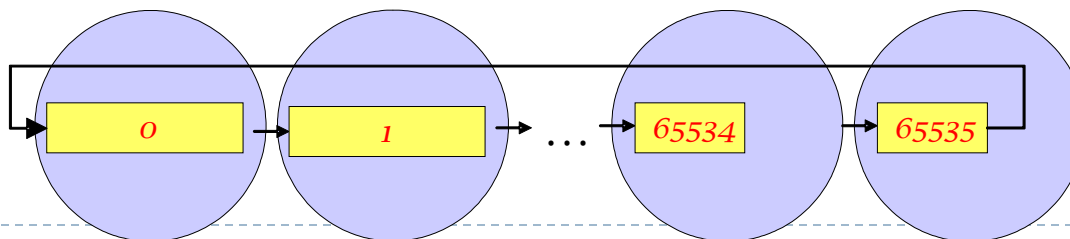
▶ 규칙성이 있음

- 수도 계량기나 자동차의 주행거리계와 비슷하게 동작
- 한계를 벗어나면 다시 처음으로 돌아가서 시작

▶ 각 자료형의 최대값과 최소값은 limits.h에 정의



short의 경우



unsigned short의 경우

예제 : 오버플로우(overflow.c)

```
#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\n", s_money);
```

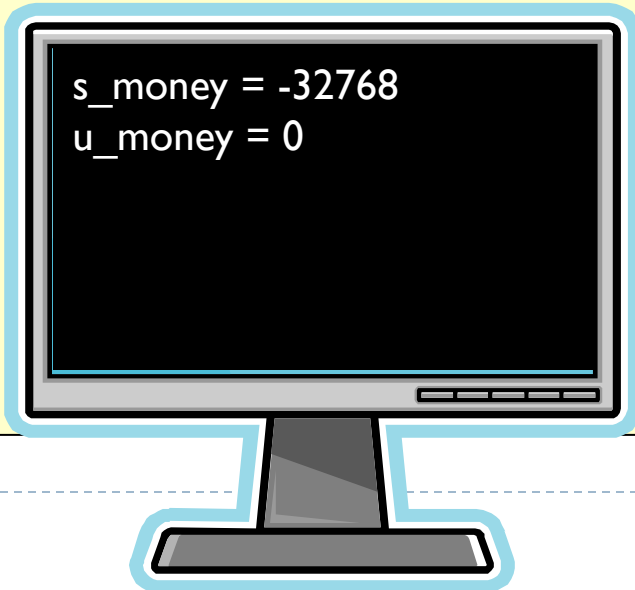
```
    u_money = u_money + 1;
```

```
    printf("u_money = %d\n", u_money);
```

```
    return 0;
```

```
}
```

오버플로우 발생!!



```
s_money = -32768
u_money = 0
```

정수 상수[1/2]

- ▶ 정수 상수는 12나 100과 같이 숫자로 표시
 - 숫자를 적으면 기본적으로 int형이 됨
 - ▶ `sum = 123;` // 123은 int형
- ▶ 상수의 자료형을 명시하려면 상수 뒤에 접미사 추가
 - `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

정수 상수[2/2]

- ▶ 정수 상수는 10진법 이외의 진법으로도 표기 가능

- 8진법 상수 : 0~7까지의 숫자로 구성됨

- ▶ 10진수 10을 8진수로 표현한다면

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

- 16진법 상수 : 0~9, A~F까지의 숫자와 문자로 구성됨

- ▶ 10진수 10을 16진수로 표현한다면

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

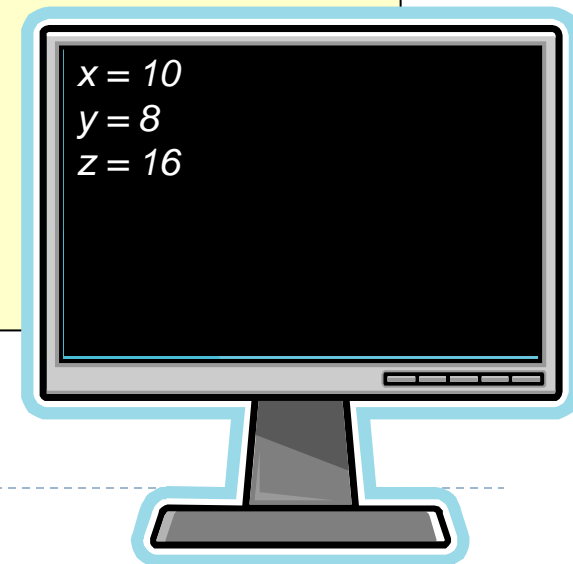
```
int x = 10;  
int y = 012;      // 8진수  
int z = 0xA;      // 16진수
```

자릿수증가

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(int_const.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; // 0x10은 16진수이고 int형이고 값은 십진수로 16이다.  
  
    printf("x = %d\n", x);  
    printf("y = %d\n", y);  
    printf("z = %d\n", z);  
  
    return 0;  
}
```

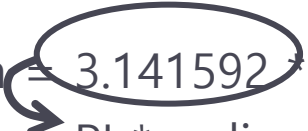



기호 상수[1/2]

▶ 기호 상수(symbolic constant)

- 상수에 이름을 붙이는 방법
- 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 높아짐
- 값을 쉽게 변경 가능

기호 상수[2/2]

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

```
int main(void)
```

```
{
```

```
...
```

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

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

```
...
```

```
}
```

1050

1050

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;
```

```
...
```

```
}
```

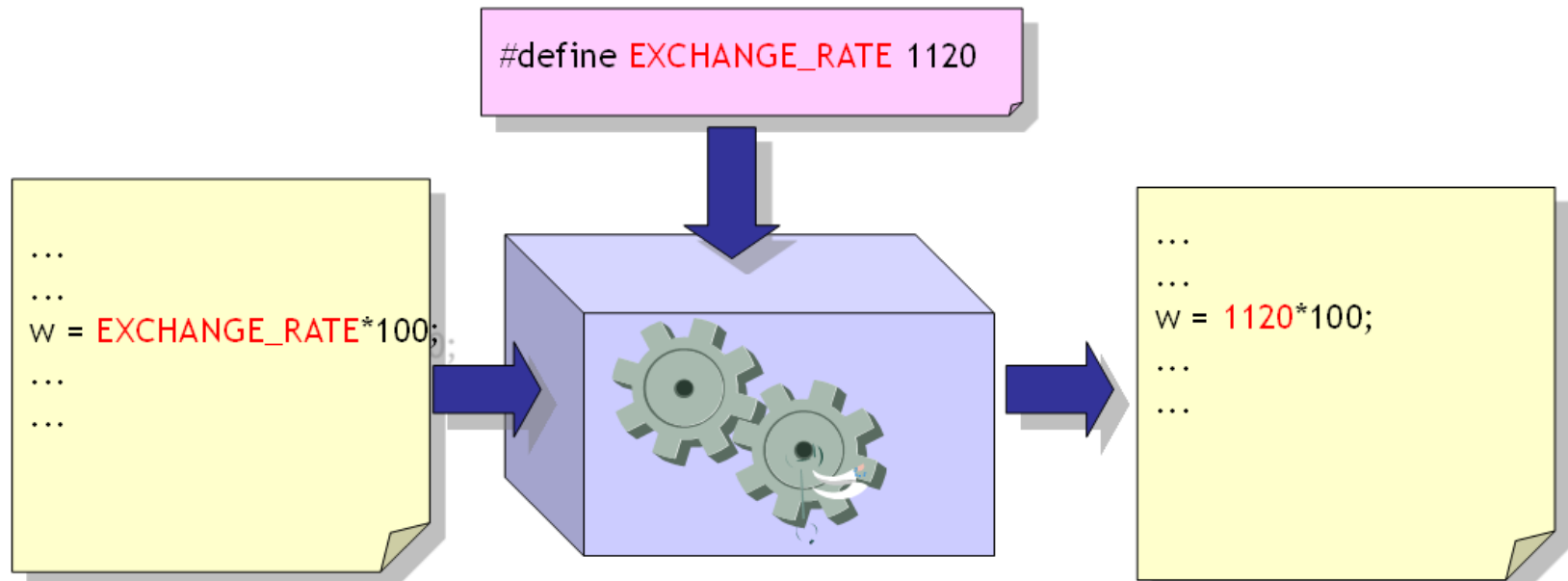
1050

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

기호 상수 만드는 방법[1/2]

EXCHANGE_RATE이라는 기호를 1120으로 정의

#define EXCHANGE_RATE 1120

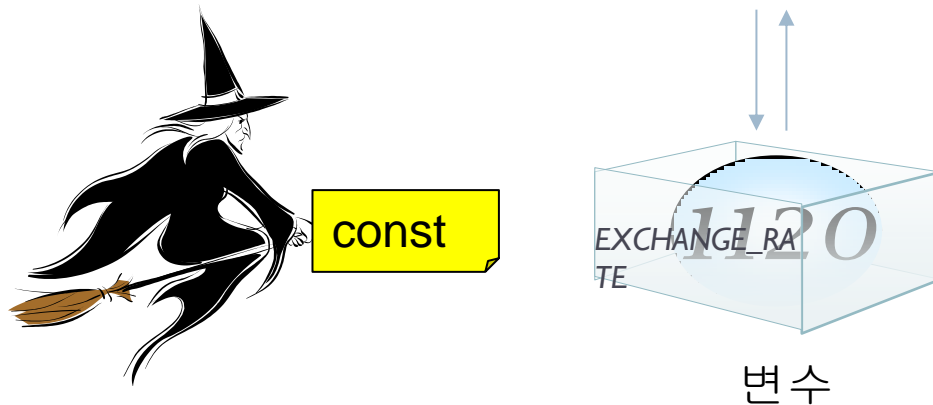


전처리기

기호 상수 만드는 방법 [2/2]

변수가 값을 변경할 수 없게 한다.

const int EXCHANGE_RATE = 1120;



예제 : 기호 상수(sym_const.c)

```
#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);
```

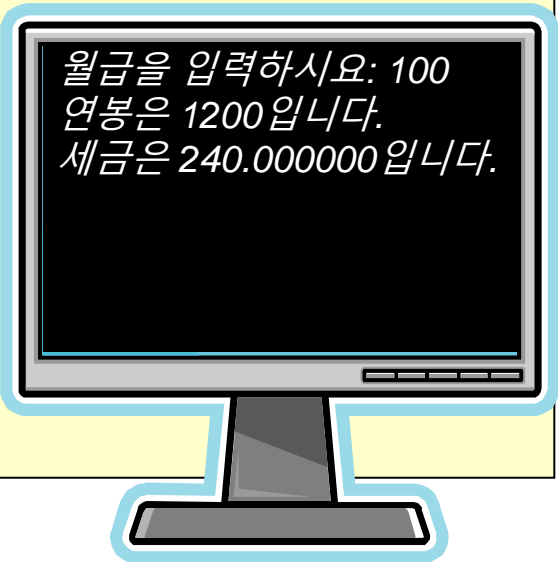
```
    y_salary = MONTHS * m_salary;     // 순수입 계산
```

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

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

```
    return 0;
```

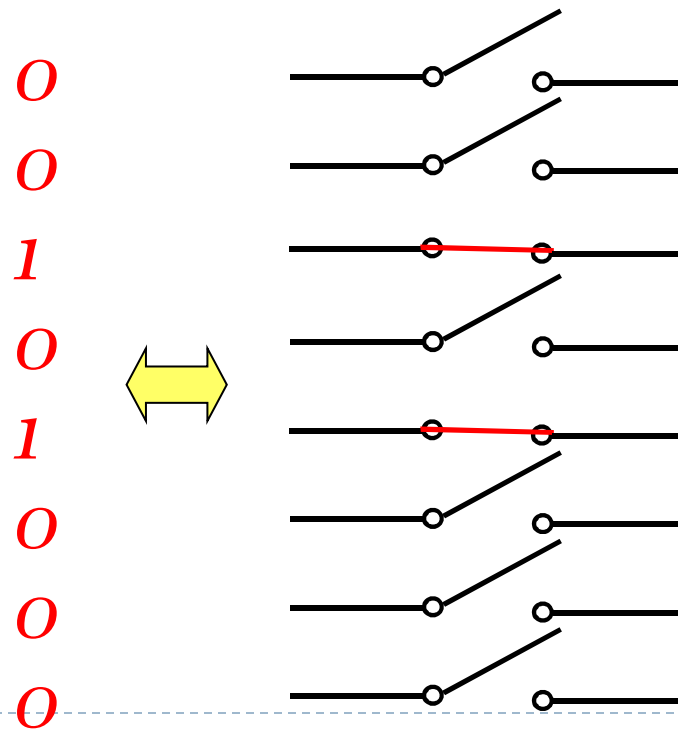
```
}
```



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

정수 표현 방법[1/2]

- ▶ 컴퓨터에서 정수는 이진수 형태로 표현
 - 이진수의 하나의 자릿수를 비트(bit)라고 함
 - ▶ 두 개의 비트로는 0, 1, 2, 3을 나타낼 수 있음(00, 01, 10, 11)
 - 이진수는 전자 스위치로 표현



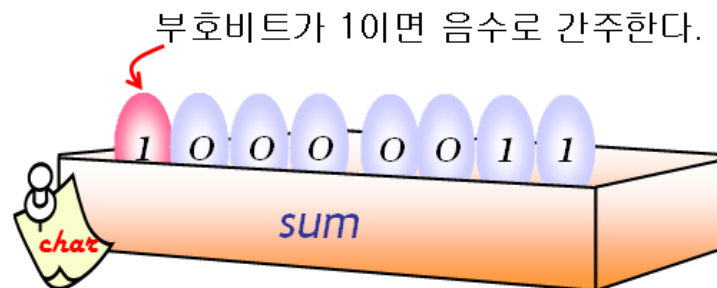
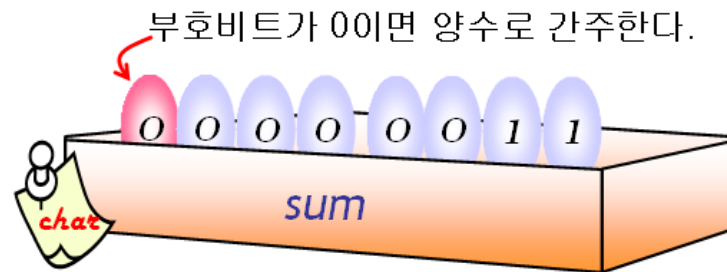
정수 표현 방법 [2/2]

▶ 양수

- 십진수를 이진수로 변환하여 저장

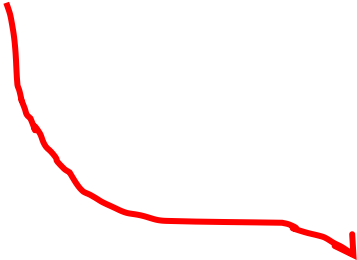
▶ 음수

- 보통은 첫 번째 비트를 부호 비트(sign bit)로 사용
 - ▶ 첫 번째 비트가 0이면 양의 정수, 1이면 음의 정수를 의미
 - ▶ 문제점 발생!



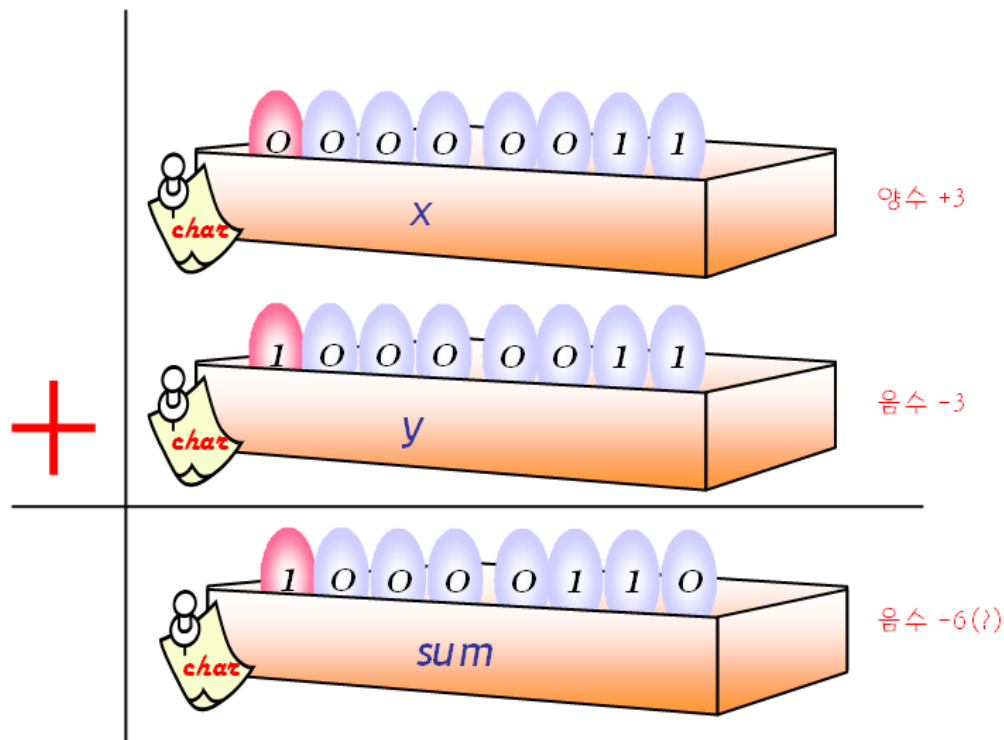
덧셈만 하는 컴퓨터

- ▶ 컴퓨터는 회로의 크기를 줄이기 위하여 덧셈 회로만 존재
- ▶ 뺄셈은 다음과 같이 덧셈으로 변환


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

음수를 표현하는 첫 번째 방법

- ▶ 맨 처음 비트를 부호 비트로 간주하는 방법
 - 양수와 음수의 덧셈 연산을 하였을 경우, 결과 부정확
 - ▶ (예) $+3 + (-3) = -6$ (X)

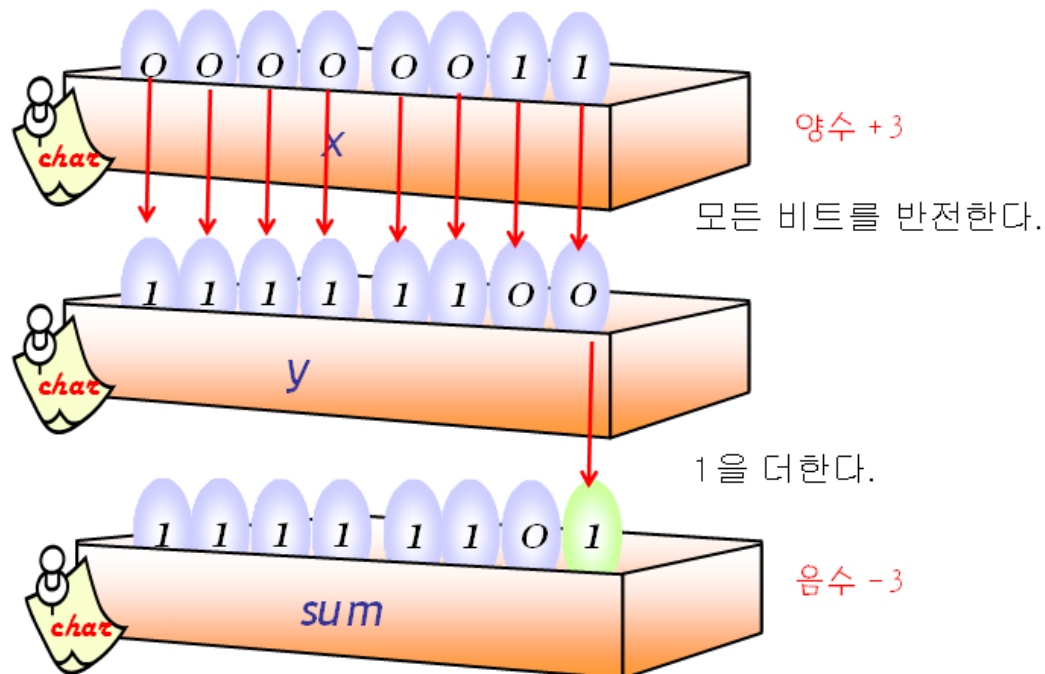


이 방법으로 표현된 이진수를 평범하게 더하면 결과가 부정확합니다.



음수를 표현하는 두 번째 방법

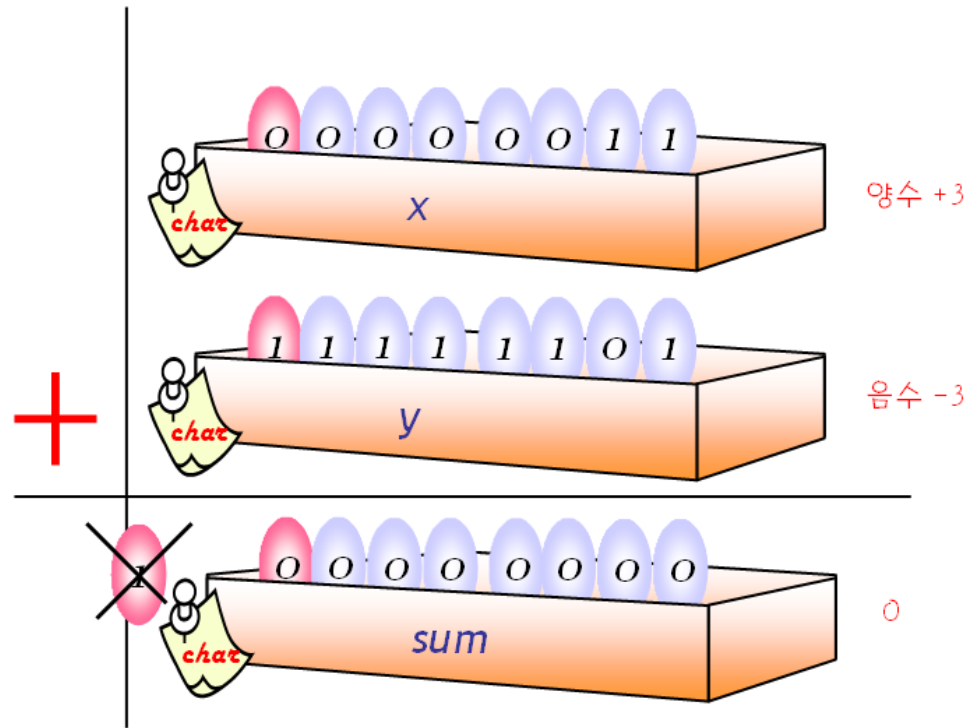
- ▶ 2의 보수로 음수를 표현
 - 표준적인 음수 표현 방법
- ▶ 2의 보수를 만드는 방법
 - 이진수의 각 비트를 반전(1을 0으로, 0을 1로)시킨 후 1을 더함



2의 보수의 덧셈

▶ 2의 보수 사용

- 부호 비트에 신경 쓸 필요 없이 어떤 부호든지 이진수 덧셈으로 계산 가능



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



예제 : 2의 보수(two_compl.c)

```
/* 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
```

```
y = FFFFFFFD
```

```
x+y = 00000000
```



부동 소수점형 [1/4]

- ▶ 컴퓨터에서 실수는 부동 소수점형으로 표현
 - 소수점이 떠서 움직인다는 의미
 - 과학자들이 많이 사용하는 과학적 표기법과 유사

실수의 정밀도를 나타낸다.

실수의 표현범위를 나타낸다.

$$1.49598 \times 10^8$$

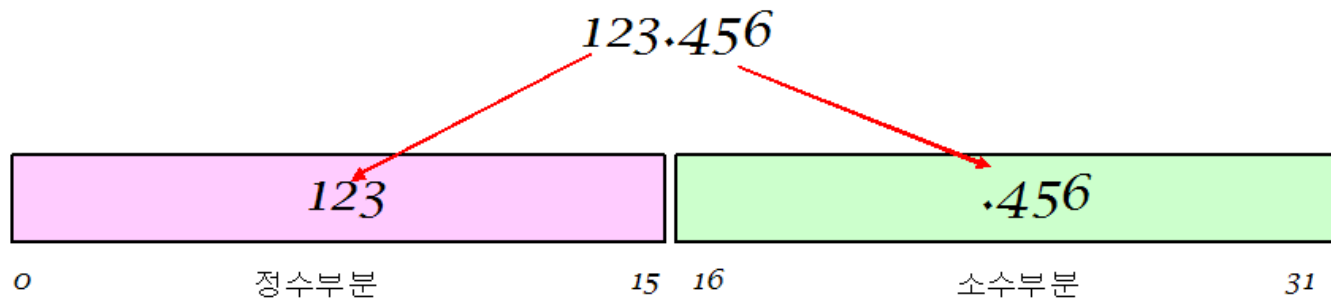
가수부분 지수부분

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

부동 소수점 형[2/4]

▶ #1 고정 소수점 표현 방식

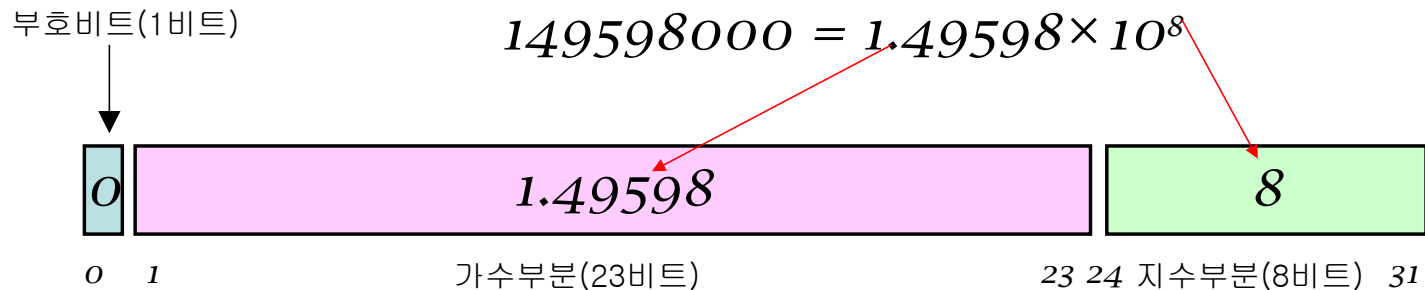
- 정수 부분을 위하여 일정 비트를 할당하고 소수 부분을 위하여 일정 비트를 할당
- 전체가 32비트이면 정수 부분 16비트, 소수 부분 16비트 할당
- 과학과 공학에서 필요한 아주 큰 수를 표현할 수 없음



부동 소수점 형 [3/4]

▶ #2 부동 소수점 표현 방식

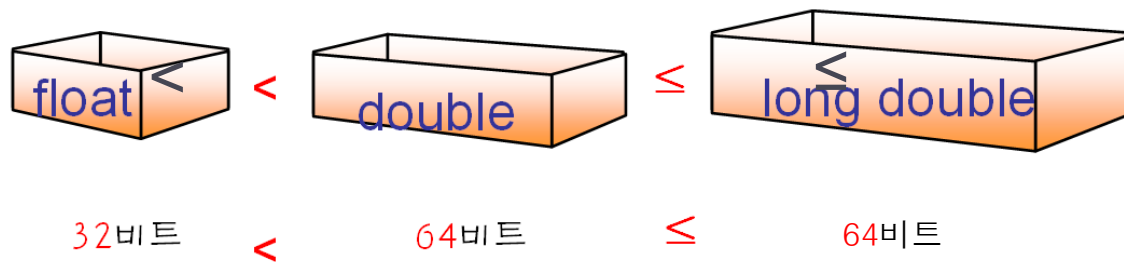
- 실수를 가수와 지수 부분으로 나누어서 표현
- 표현할 수 있는 범위가 대폭 늘어남
- 10^{-38} 에서 10^{+38}
- 부동 소수점 방식은 고정 소수점 방식보다 계산 속도가 느림
 - ▶ 부동 소수점 장치가 CPU에 포함되어 있는 경우가 많음



부동 소수점형 [4/4]

▶ float, double, long double의 부동 소수점 자료형

- float
 - ▶ 32비트 중 8비트를 지수에 할당하고 나머지 24비트가 가수에 할당
 - ▶ 유효 숫자 6자리 까지만 나타냄(예, 0.123456)
- double, long double
 - ▶ 64비트 중 11비트 정도를 지수에 할당하고 나머지 53비트가 가수에 할당
 - ▶ 정확한 숫자는 컴퓨터 시스템에 따라 달라지지만 대략 유효 숫자 16자리를 나타냄



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	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{+308}$
long double	부동소수점		

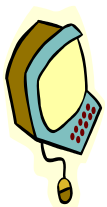
예제 : 부동 소수점 형(floating.c)

```
/* 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("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.
- 2e+10 // +나 -기호를 지수부에 붙일 수 있다.
- 9.26E3 // 9.26×10^3
- 0.67e-9 // 0.67×10^{-9}

부동 소수점 오버플로우

▶ 오버플로우

- 변수에 대입된 수가 너무 커서 변수가 저장할 수 없는 상황을 의미
- float형 변수는 약 1×10^{38} 이상을 넘는 수는 저장 못함

```
#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

▶ 언더플로우(underflow)

- 오버플로우와 반대상황
- 부동 소수점 수가 너무 작아서 표현하기가 힘든 상황

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

```
int main(void)
```

```
{
```

```
    float x = 1.23456e-38;
```

```
    float y = 1.23456e-40;
```

```
    float z = 1.23456e-46;
```

숫자가 작아서 언더플로우 발생

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

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

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

```
}
```

```
x = 1.234560e-038
```

```
y = 1.234558e-040
```

```
z = 0.000000e+000
```



부동 소수점형 사용시 주의사항

- ▶ 오차가 있을 수 있음
 - 오차를 줄려면 float보다 double 사용
 - $1.0e20 + 5.0$ 을 저장하려면 유효 숫자가 적어도 20개 필요
 - ▶ 100000000000000000005를 저장하기 위한 유효 숫자 공간을 가지고 있지 않음

```
#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
```

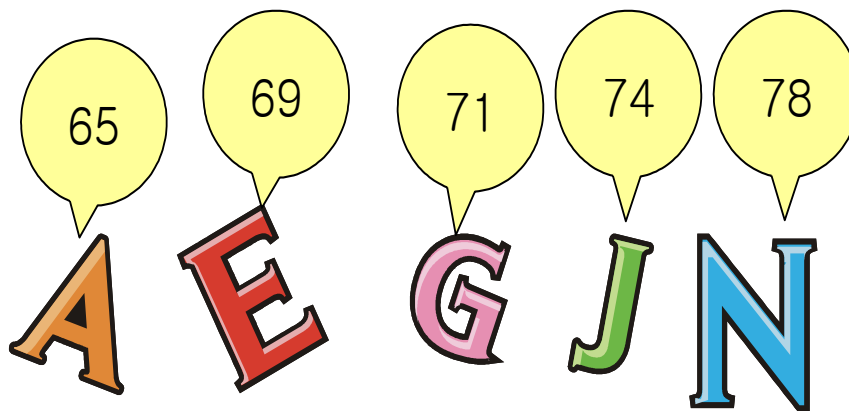


중간 점검

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

문자형 [1/2]

- ▶ 문자
 - 한글이나 영어에서의 하나의 글자, 숫자, 기호 등을 의미
- ▶ 문자는 컴퓨터보다는 인간에게 중요
- ▶ 컴퓨터는 문자도 숫자를 이용하여 표현



문자형 [2/2]

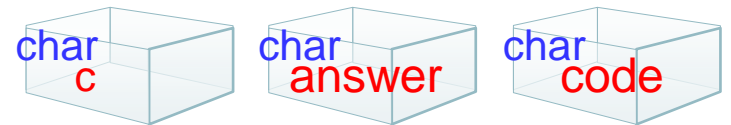
- ▶ 모든 사람들이 문자 데이터를 교환하려면 공통적인 규격 필요
- ▶ 아스키 코드(**ASCII**: American Standard Code for Information Interchange)
 - 33개의 제어 문자 코드와 95개의 문자 코드로 구성
 - 8비트를 사용하여 영어 알파벳 표현
 - (예) !는 33, 'A'는 65, 'B'는 66, 'a'는 97, 'b'는 98

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


문자 변수

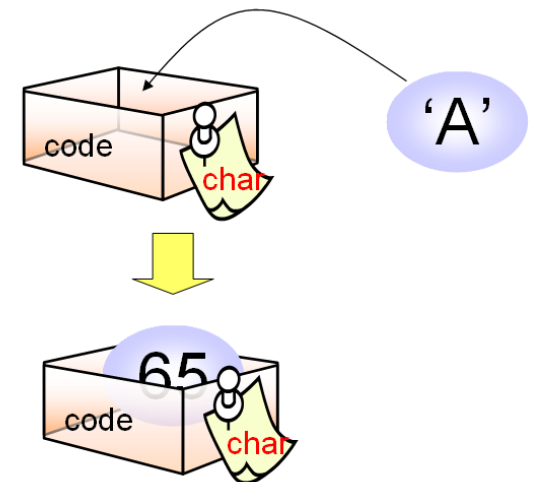
- ▶ 정수를 저장할 수 있는 자료형은 문자 저장 가능(문자가 정수로 표현)
 - 0~127까지의 숫자만 이용하므로 8비트로 충분
 - char형은 8비트 정수 저장 가능
- ▶ char형의 변수가 문자 저장

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



- ▶ 문자를 저장하려면 아스키 코드 값을 대입

```
code = 65;           // 'A' 저장  
code = 'A';
```



예제 : 문자 변수와 문자 상수(char_var.c)

- ▶ 형식 지정자를 %c로 해서 실행해 본 후 %d로 변경해서 결과 비교

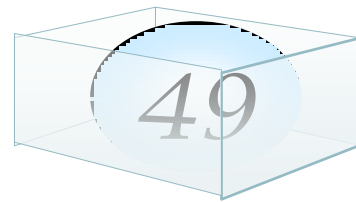
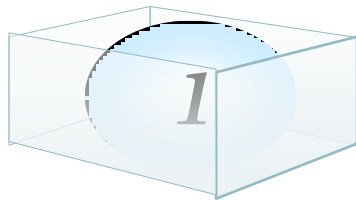
```
/* 문자 변수와 문자 상수*/  
#include <stdio.h>  
  
int main(void)  
{  
    char code1 = 'A';    // 문자 상수로 초기화  
    char code2 = 65;     // 아스키 코드로 초기화  
  
    printf("문자 상수 초기화 = %c\n", code1);  
    printf("아스키 코드 초기화 = %c\n", code2);  
}
```

문자 상수 초기화 = A
아스키 코드 초기화 = A



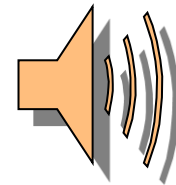
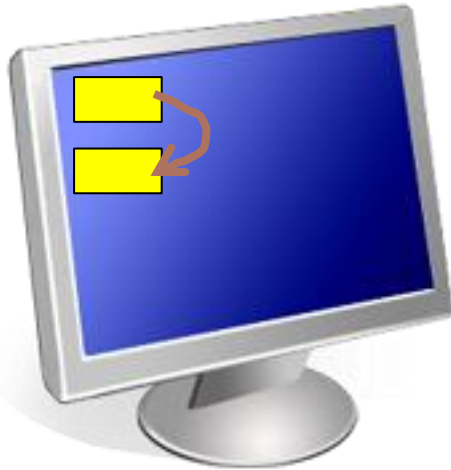
Quiz

(Q) 1과 '1'의 차이점은?



제어 문자

- ▶ 아스키 코드표에는 제어 문자들도 함께 정의
 - 인쇄 목적이 아니라 제어 목적으로 사용되는 문자들
 - ▶ (예) 줄바꿈 문자, 탭 문자, 벨소리 문자, 백스페이스 문자

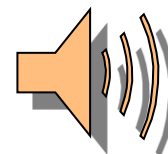


제어 문자를 나타내는 방법

▶ 아스키 코드를 직접 사용

- 가장 쉬운 방법이지만 아스키 코드값을 암기해야 함

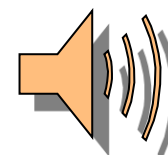
```
char beep = 7;  
printf("%c", beep);
```



▶ 이스케이프 시퀀스 사용

- 역슬래시(\)와 의미를 나타내는 한 글자를 붙여서 기술

```
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 : 이스케이프 시퀀스(escape.c)

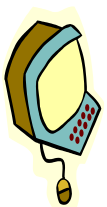
```
#include <stdio.h>
int main()
{
    int id, pass;

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

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

    printf("pass: ____\b\b\b\b\b");
    scanf("%d", &pass);
    printf("a입력된 아이디는 \b"%d\b"이고 패스워드는 \b"%d\b"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 있음
 - signed char형은 -128 ~ +127 까지의 정수 저장
 - unsigned char형은 0 ~ 255 까지의 정수 저장

```
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 67A B C

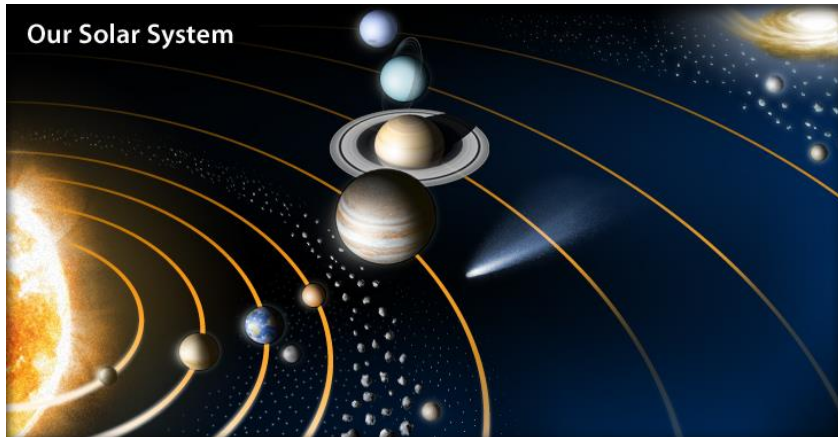


중간 점검

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

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.



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 8.311111초

도전 문제

- ▶ 위의 프로그램의 출력은 8.31111...초로 나온다. 이것을 분과 초로 나누어서 8분 20초와 같은 식으로 출력하도록 변경하라. 필요하다면 형변환을 사용하라. 추가적인 정수 변수를 사용하여도 좋다.