

# C 언어 EXPRESS(개정3판)



## 제 4장 변수와 자료형



# 이번 장에서 학습할 내용

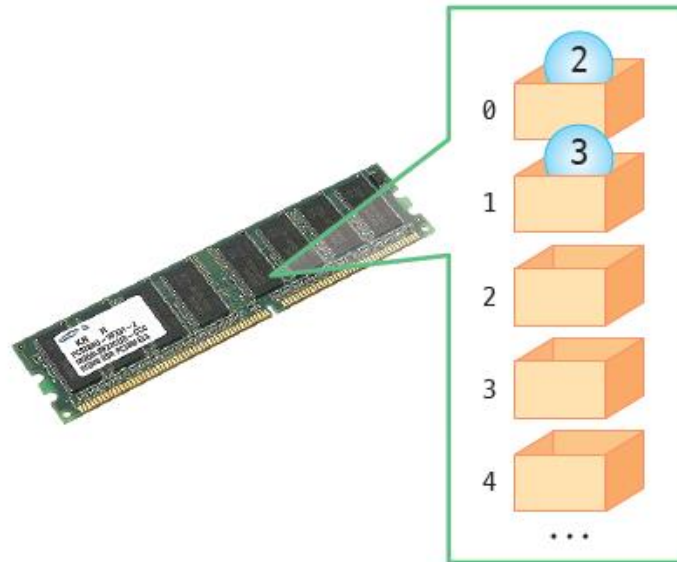


- \* 변수와 상수의 개념 이해
- \* 자료형
- \* 정수형
- \* 실수형
- \* 문자형
- \* 기호 상수 사용
- \* 오버플로우와 언더플로우 이해



# 변수는 어디에 만들어지는가?

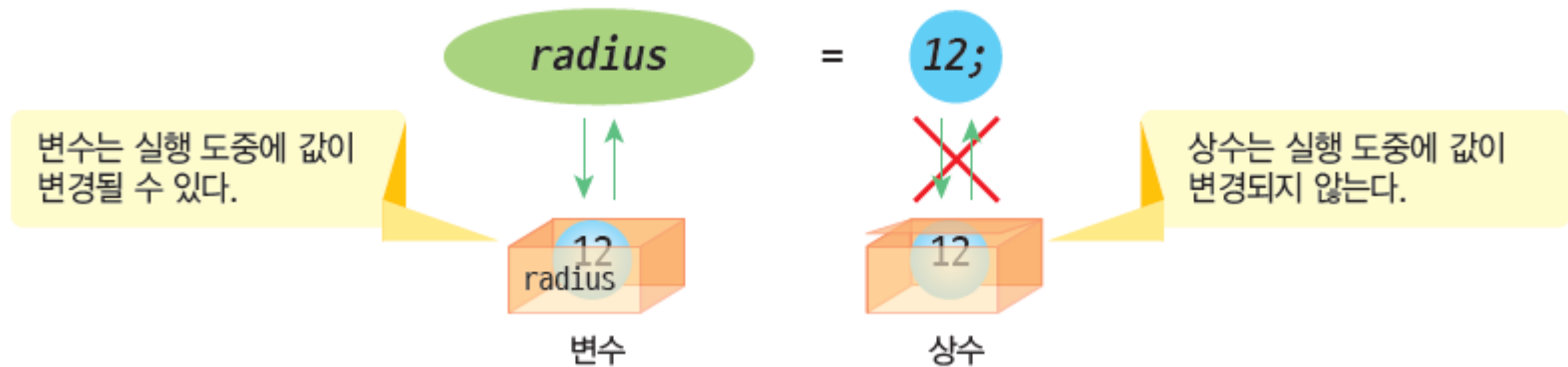
- 변수는 바로 메인 메모리(main memory)에 만들어진다.
- 우리는 변수 이름을 사용하여 메모리 공간을 사용하게 된다





# 변수와 상수

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



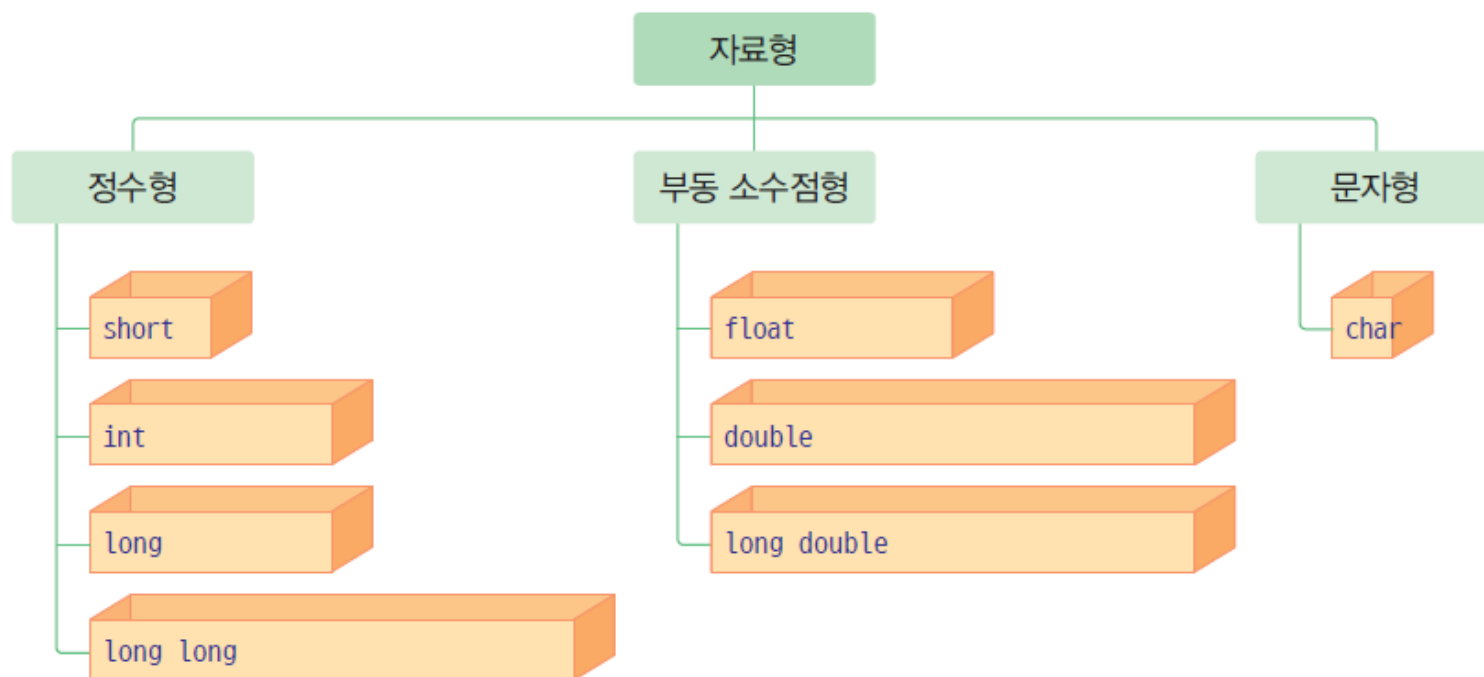


# 자료형

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



# 자료형의 분류

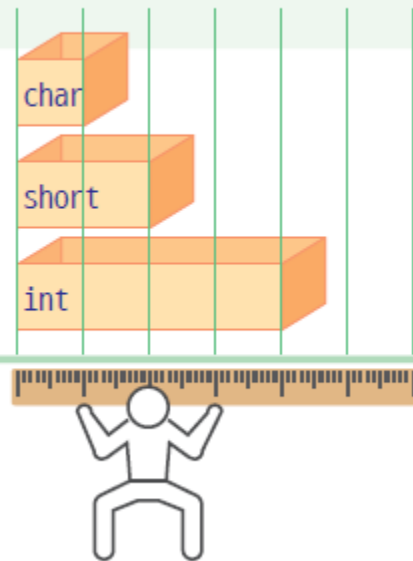




# 자료형의 크기

Syntax: sizeof()

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



sizeof 연산자는 변수나  
데이터 타입의 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));

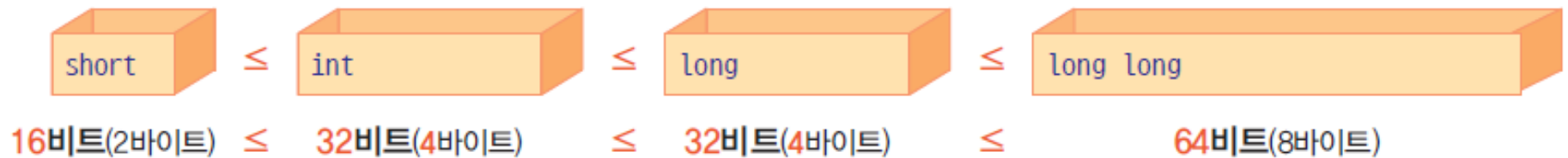
    return 0;
}
```





# 정수형

- short형
- int형
- long형
- long long형





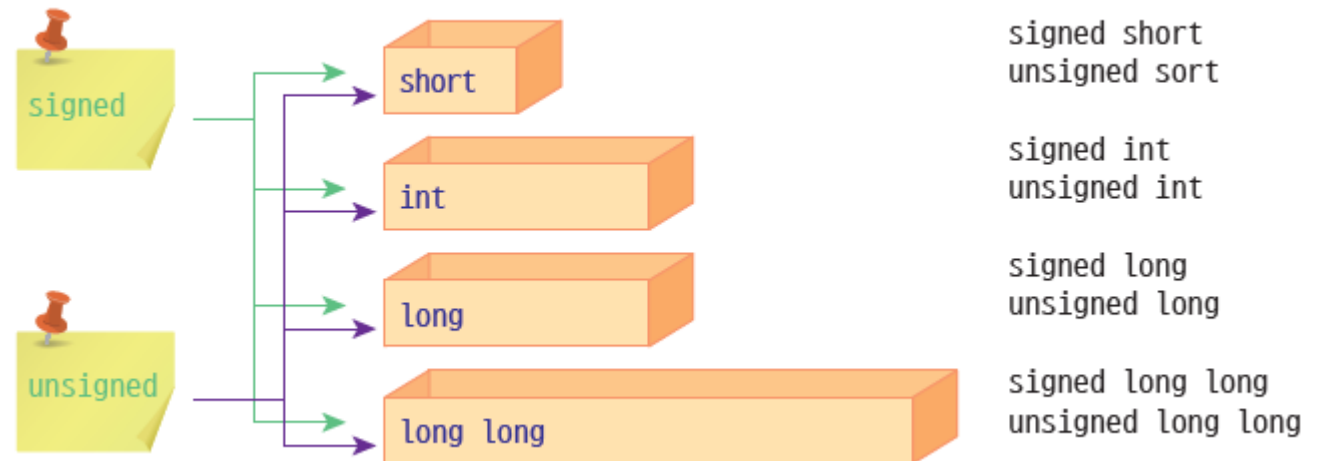
# 정수형

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



# signed, unsigned 수식자

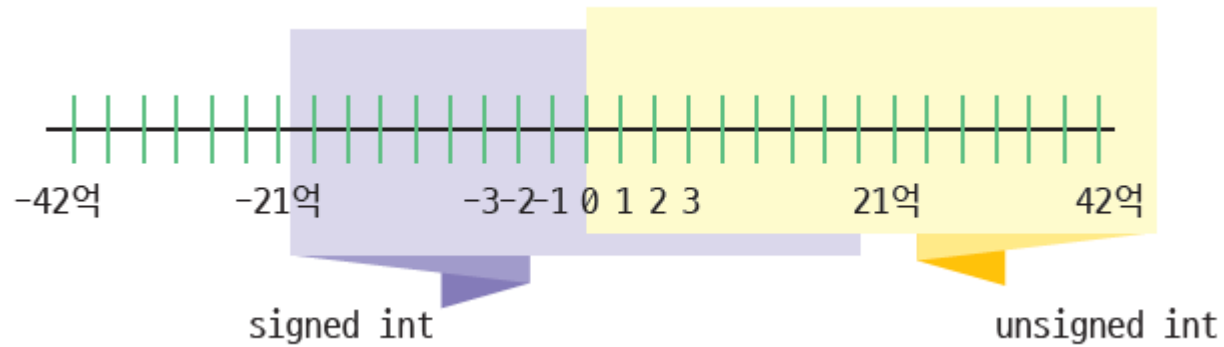
- unsigned
  - 음수가 아닌 값만을 나타냄을 의미
  - unsigned int
- signed
  - 부호를 가지는 값을 나타냄을 의미
  - 흔히 생략





# unsigned int

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





# unsigned 수식자

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



# 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);    오버플로우 발생!!
    return 0;
}
```

```
S_money = -32768
U_money = 0
```



# 오버플로

- 규칙성이 있다.
  - 수도 계량기나 자동차의 주행거리계와 비슷하게 동작



short의 경우



unsigned short의 경우



# 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





# 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

z = 16

x = 10

y = 8

z = 16



# 2진법, 10진법, 8진법, 16진법

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

```
int main(void)
```

```
{
```

```
+
```

```
/* ... */
```

```
int a = 0b1010;
```

```
int b = 012;
```

```
int c = 0xA;
```

```
printf("%d\n", a);
```

```
printf("%#o\n", a);
```

```
printf("%#X\n", a);
```

```
printf("%d\n", b);
```

```
printf("%#o\n", b);
```

```
printf("%#X\n", b);
```

```
printf("%d\n", c);
```

```
printf("%#o\n", c);
```

```
printf("%#X\n", c);
```

```
return 0;
```

```
}
```



# 2진법, 10진법, 8진법, 16진법

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

```
int main(void)
```

```
{
```

```
    int a = 10, i;
```

```
    printf("%d\n", a);
```

```
    printf("%#o\n", a);
```

```
    printf("%#X\n", a);
```

```
    for (i=a; i >= 0; i--) {
```

```
        printf("%d", (a>>i)&1);
```

```
    }
```

```
    return 0;
```

```
}
```



## 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 = 1100 * dollar1;
```

```
won2 = 1100 * dollar2;
```

```
...
```

```
}
```

1050

1050

리터럴 상수를 사용하는 경우:  
등장하는 모든 곳을 수정하여야 한다.

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

```
#define EXCHANGE_RATE 1100
```

```
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`

기호 상수                  값

Syntax: 기호상수선언

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

기호 상수                  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;
```

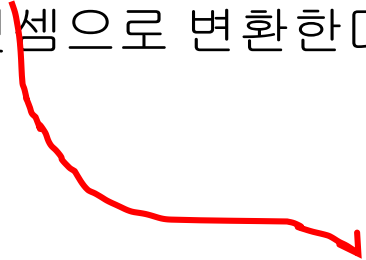
```
}
```

```
월급을 입력하시요: 200
연봉은 2400입니다.
세금은 480.000000입니다.
```



# 컴퓨터는 덧셈만 할 수 있다

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

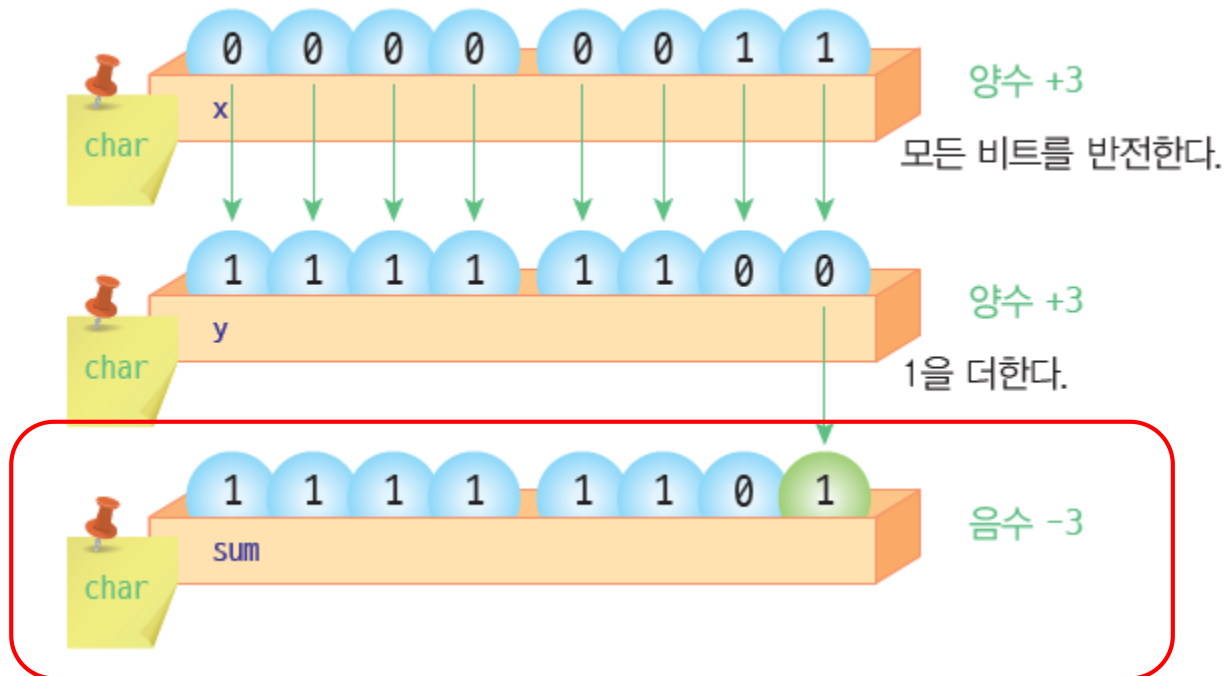

$$3-3 = 3+(-3)$$





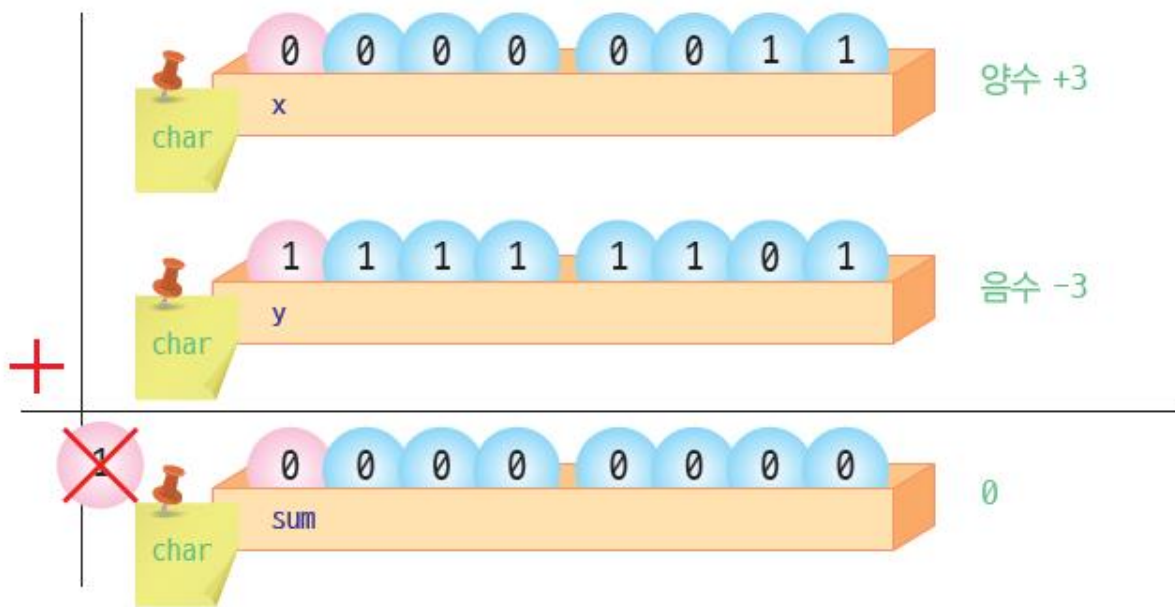
# 음수를 표현하는 방법 2의 보수

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





# 2의 보수로 양수와 음수를 더하면



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





# 2의 보수 정리

비트 패턴	정수	비고
00000000	0	양의 정수
00000001	1	
00000010	2	
00000011	3	
...	...	
01111111	127	
10000000	-128	음의 정수
10000001	-127	
10000010	-126	
...	...	
11111101	-3	
11111110	-2	
11111111	-1	

비트 반전

1을 더한다

11111100



# 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



# 부동소수점형

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

실수의 정밀도를 나타낸다.

실수의 표현 범위를 나타낸다.

$$\underbrace{1.49598}_{\text{가수 부분}} \times \underbrace{10^8}_{\text{지수 부분}}$$

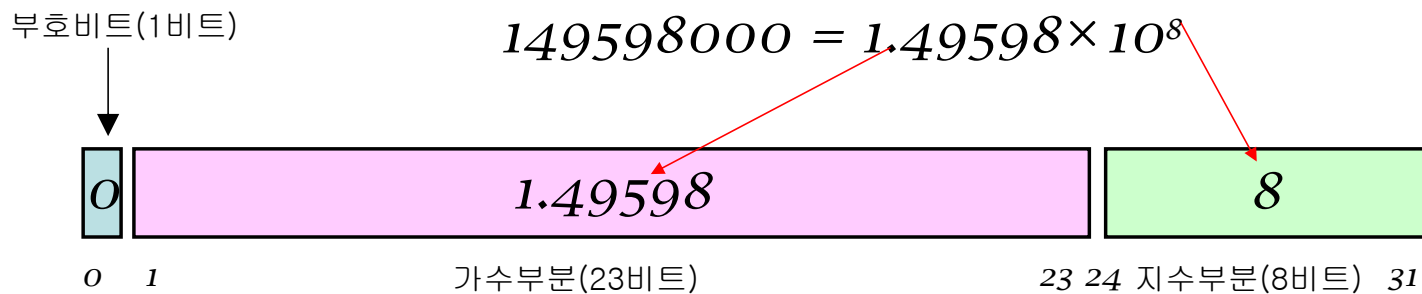


149,598,000



# 실수를 표현하는 방법

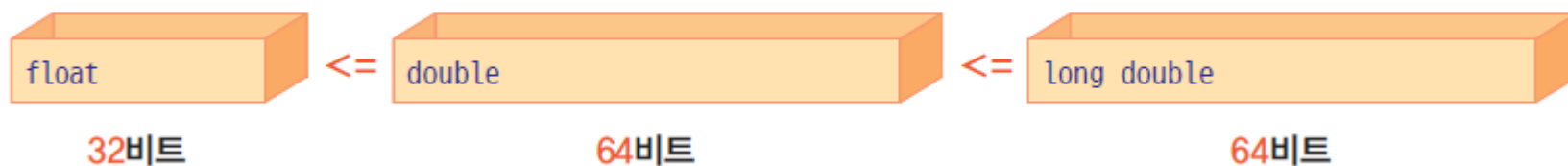
## ● #2 부동 소수점 방식



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



# 부동 소수점 형



소수 8자리 까지 정확  
출력시 %lf 또는 %f 사용

소수 16자리 까지 정확  
출력시 %lf

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 = 1.23456788063049320000000000  
y = 1.23456789012345670000000000
```



# 부동 소수점 상수

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

1.23456

2. // 소수점만 붙여도 된다.

.28 // 정수부가 없어도 된다.

2e+10 // +나 -기호를 지수부에 붙일 수 있다.

9.26E3 //  $9.26 \times 10^3$

0.67e-7 //  $0.67 \times 10^{-9}$



# 부동 소수점 오버플로우

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

```
int main(void)
```

```
{
```

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

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

```
}
```

숫자가 커서 오버플로우

발생

1e38까지 가능

inf : 무한대

```
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);
```

```
}
```

범위를 벗어나면 값이  
정확하지 않다

숫자가 작아서  
언더플로우 발생

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



# 부동소수점 연산 시 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.

20개의 0에 5를 더해 정  
확하게 표현하지 못하므  
로 잘린다.

0.000000



# 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
0	0	NULL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOL
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	HT
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3

Dec	Hex	문자
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	space
33	21	!
34	22	"
35	23	#
36	24	\$
37	25	%
38	26	&
39	27	'

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	;

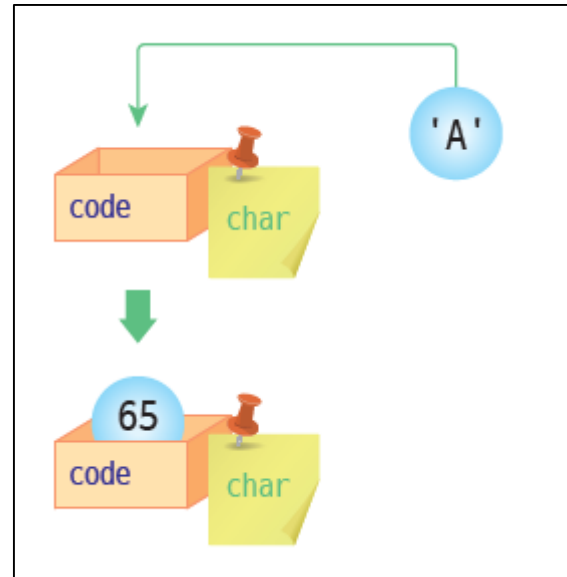
Dec	Hex	문자
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O



# 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);    %C : 문자출력  
    printf("code2 = %c\n", code2);    %d : 정수출력  
}
```

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



# A~Z까지 자동출력하기

```
#include<stdio.h>
int main(void)
{
    char c = 'A';

    for (c; c < 'Z'; c++)
    {
        printf("%C ----> %d Wn", c, c);
    }
    return 0;
}
```



# 제어 문자 / escape sequence / 리터럴 문자

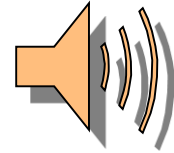
제어 문자	이름	의미
\0	널문자	
\a	경고(bell)	"삐"하는 경고음 발생
\b	백스페이스(backspace)	커서를 현재의 위치에서 한 글자 뒤로 옮긴다.
\t	수평탭(horizontal tab)	커서의 위치를 현재 라인에서 설정된 다음 탭 위치로 옮긴다.
\n	줄바꿈(newline)	커서를 다음 라인의 시작 위치로 옮긴다.
\v	수직탭(vertical tab)	설정되어 있는 다음 수직 탭 위치로 커서를 이동
\f	폼피드(form feed)	주로 프린터에서 강제로 다음 페이지로 넘길 때 사용된다.
\r	캐리지 리턴(carriage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



# 제어 문자를 나타내는 방법

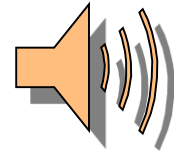
- 아스키 코드를 직접 사용

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



- 이스케이프 시퀀스 사용

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

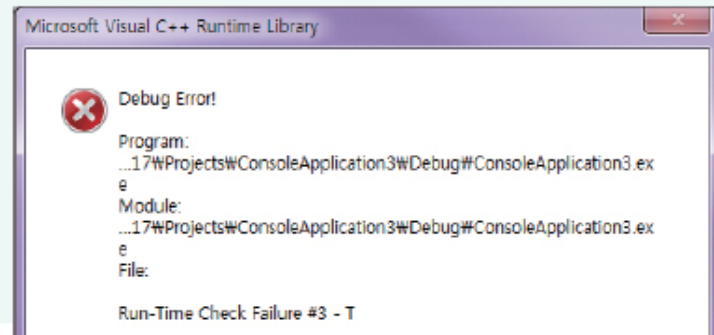
```
65 66 67
A B C
```



# Lab: 변수의 초기값

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 }
```

변수는 사용하기 전에 반드시 초기화  
시켜야 함!

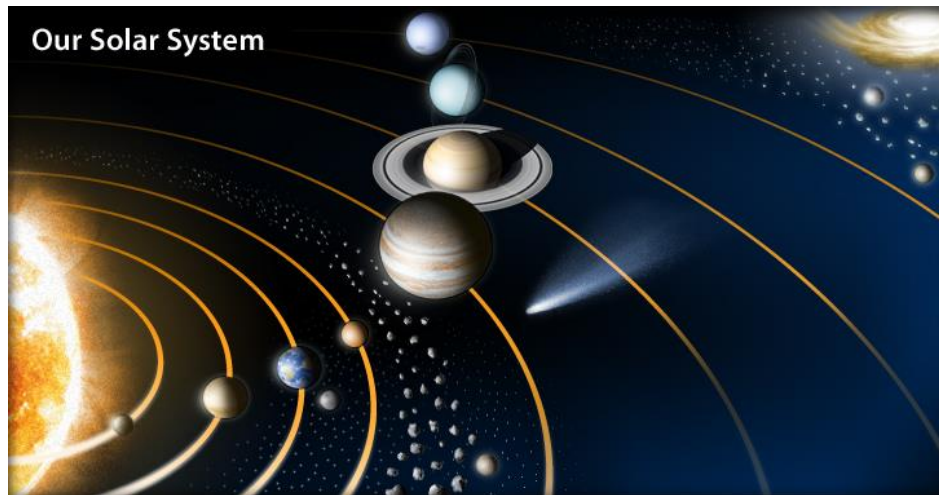
## 실행결과

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.





## 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  $(\text{도달 시간} = \text{거리} / (\text{빛의 속도}))$ 으로 계산할 수 있다.
- 실수형을 `printf()`로 출력할 때는 `%f`나 `%lf`를 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 %f초\n", time); // 시간을 출력한다.
```

```
    return 0;
```

```
}
```

```
빛의 속도는 300000.000000km/s
```

```
태양과 지구와의 거리 149600000.000000km
```

```
도달 시간은 498.666667초
```