2강. 변수와 연산자

3장. 변수와 연산자

- 연산자란 무엇인가?
  - 연산을 요구할 때 사용되는 기호
  - ex : +, -, *, /

```
int main(void)
{
        3+4;    //
        return 0;
}
```

- 변수란 무엇인가?
  - 데이터를 저장할 수 있는 메모리 공간에 붙여진 이름

- 다양한 형태(자료형)의 변수
  - 정수형 : char, int, long
  - 실수형 : float, double

- 변수의 선언 및 대입
  - 대입 연산자(=): 값을 대입하기 위한 용도의 연산자

```
int main(void)
{
    int val;        // int형 변수 val의 선언
    val = 20;       // 변수 val에 20을 저장
    . . . . . .
```

- 변수를 이용한 예제

```
#include <stdio.h>

int main(void)
{
    int a, b;                    // 쓰레기 값으로 초기화
    int c=30, d=40;

    a=10;
    b=20;

    printf("%d %d ₩n", a, b);
    printf("%d %d ₩n", c, d);
   return 0;
}
```

- 변수 선언 시 주의 사항 1
  - 변수를 함수 내에 선언할 경우, 등장 위치!

```
#include <stdio.h>

int main(void)
{
    int a;
    int b;

    a=10;
    b=20;

    printf("%d %d ₩n", a, b);
    return 0;
}
```

- 변수 선언 시 주의 사항 2
  - 첫째 : 변수의 이름은 알파벳, 숫자 언더바(_)로 구성
  - 둘째 : 대 소문자 구분
  - 셋째 : 변수의 이름은 숫자로 시작 불가, 키워드 사용 불가
  - 넷째 : 공백이 포함될 수 없음

적절치 않은 변수의 이름	적절치 않은 이유
int 7th_val	변수의 이름이 숫자로 시작
int live_inthe#	#과 같은 특수 문자는 올 수 없다.
int kor year	변수 이름에 공백이 삽입될 수 없다.

- 완성된 덧셈 프로그램

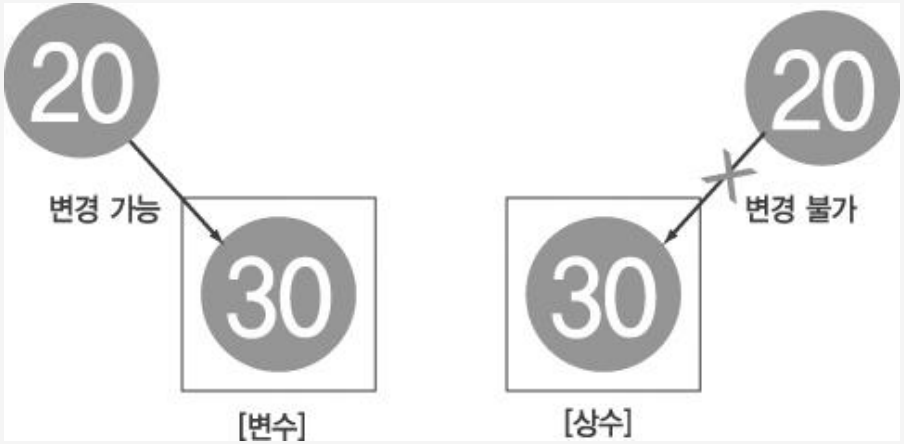
```
/* simpleadd2.c */
#include <stdio.h>

int main(void)
{
    int result;                                  //변수 선언
    result=3+4;                                  //덧셈 결과 저장

    printf("덧셈 결과 : %d ₩n", result);
    printf("%d 더하기 %d는 %d 입니다. ₩n", 3, 4, result);
    printf("변수 result에 저장된 값 : %d ₩n", result);

    return 0;
}
```

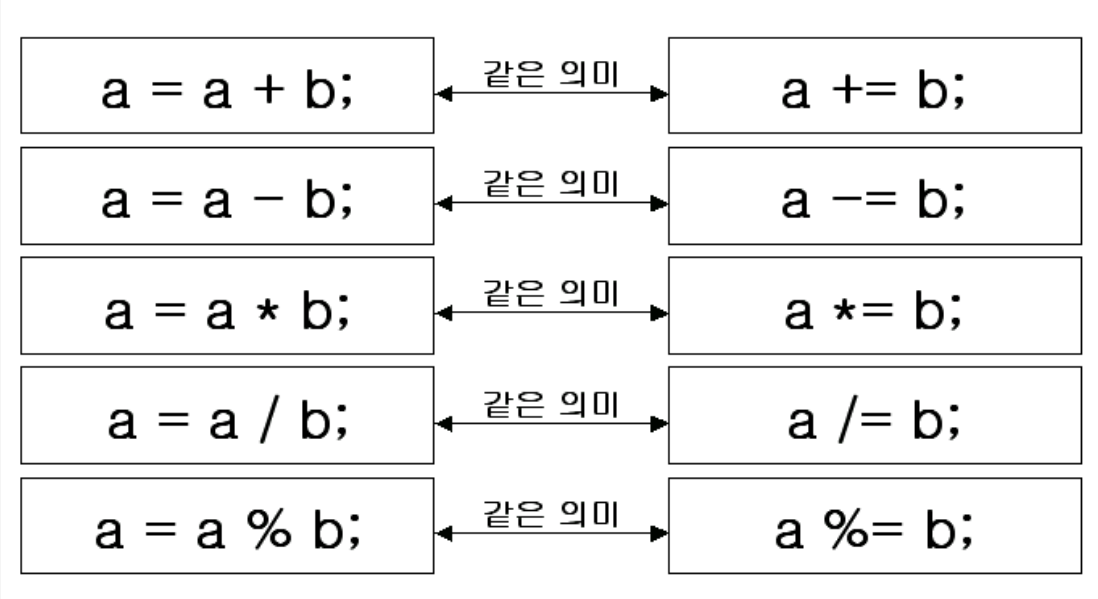
- 변수와는 다른 상수!
  - 상수도 메모리 공간을 할당 받는다.
    하지만 데이터의 변경이 불가능하다.



- 대입 연산자와 산술 연산자

연산자	연산의 예	의미	결합성
=	a=20	대입	←
+	a=4+3	덧셈	→
-	a=4-3	뺄셈	→
*	a=4*3	곱셈	→
/	a=4/3	나눗셈	→
%	a=4%3	나머지	→

- 기타 대입 연산자
  - 대입 연산자와 산술 연산자가 합해져서 다양한 형태의 대입 연산자 정의



- 부호 연산으로서 +, – 연산자
- 증가 감소 연산자

연산자	연산의 예	의미	결합성
++a	printf(“%d”, ++a)	선 증가, 후 연산	←
a++	printf(“%d”, a++)	선 연산, 후 증가	←
--b	printf(“%d”, --a)	선 감소, 후 연산	←
b--	printf(“%d”, a--)	선 연산, 후 감소	←

- 관계 연산자(비교 연산자)
  - 두 피연산자의 관계(크다, 작다 혹은 같다)를 따지는 연산자
  - true(논리적 참, 1), false(논리적 거짓, 0) 반환

연산자	연산의 예	의미	결합성
<	a<b	a가 b보다 작은가	→
>	a>b	a가 b보다 큰가	→
==	a==b	a와 b가 같은가	→
!=	a!=b	a와 b가 같지 않은가	→
<=	a<=b	a가 b보다 작거나 같은가	→
>=	a>=b	a가 b보다 크거나 같은가	→

- 논리 연산자
  - and, or, not을 표현하는 연산자
  - true(1), false(0) 반환

연산자	연산의 예	의미	결합성
&&	a&&b	true면 true 리턴	→
||	a||b	하나라도 true면 true 리턴	→
!	!a	true면 false를, false면 true 리턴	→

- 비트 단위 연산자
  - ~, &, ^, |, <<, >>

- 콤마(,) 연산자
  - 둘 이상의 변수 동시 선언 시
  - 둘 이상의 문장을 한 줄에 선언 시
  - 함수의 매개변수 전달 시

- 연산자의 우선 순위
  - 연산 순서를 결정짓는 순위

- 연산자의 결합성
  - 우선 순위가 같은 연산자들의 연산 방향

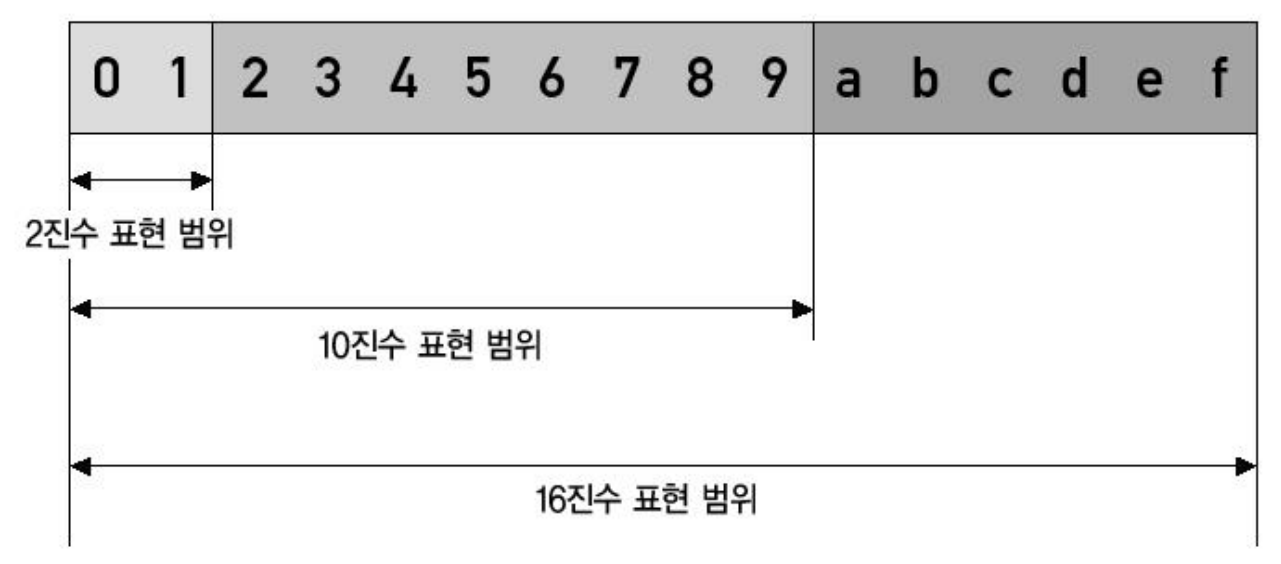
```
3+4*5/2-10
```

C 언어의 키워드들

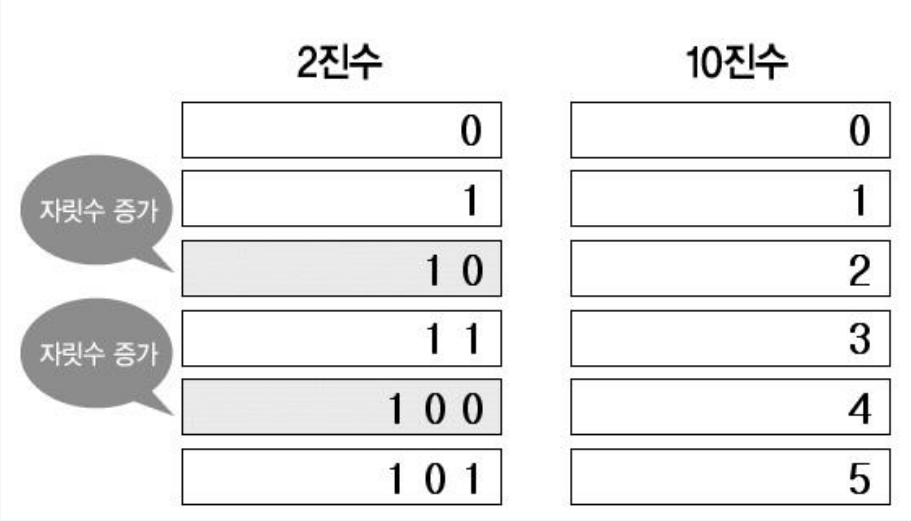
auto	double	int	struct
bool	else	long	switch
break	enum	*register*	typedef
case	extern	*restrict*	union
char	float	return	unsigned
const	for	short	void
continue	goto	signed	*volatile*
default	if	sizeof	while
do	inline	static	

4장. 데이터 표현 방식의 이해

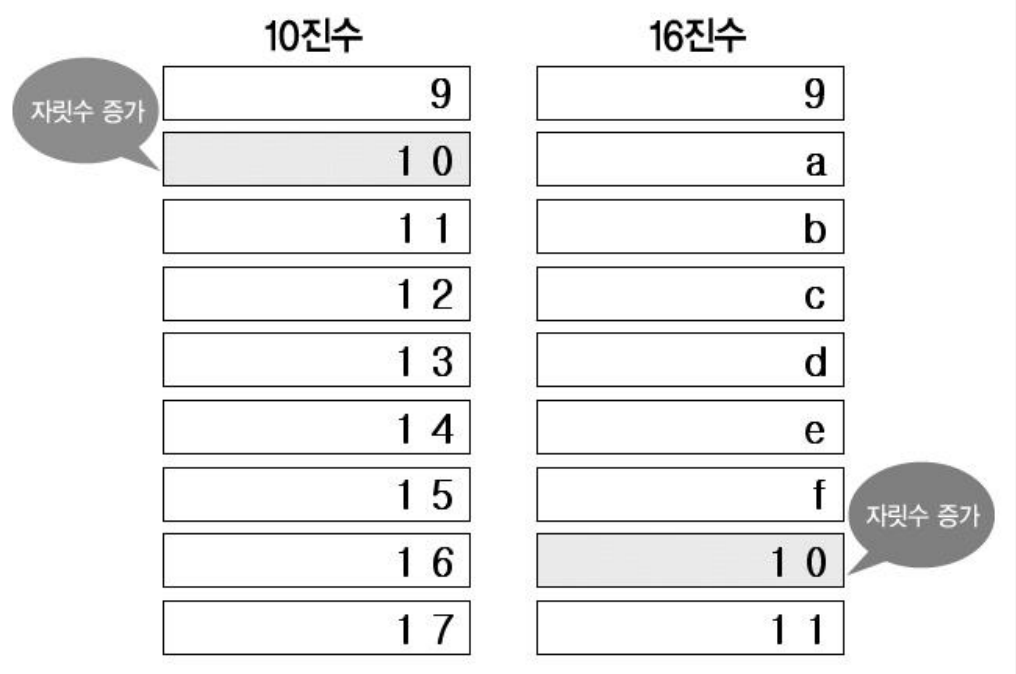
- 진법에 대한 이해
  - n 진수 표현 방식 : n개의 문자를 이용해서 데이터를 표현



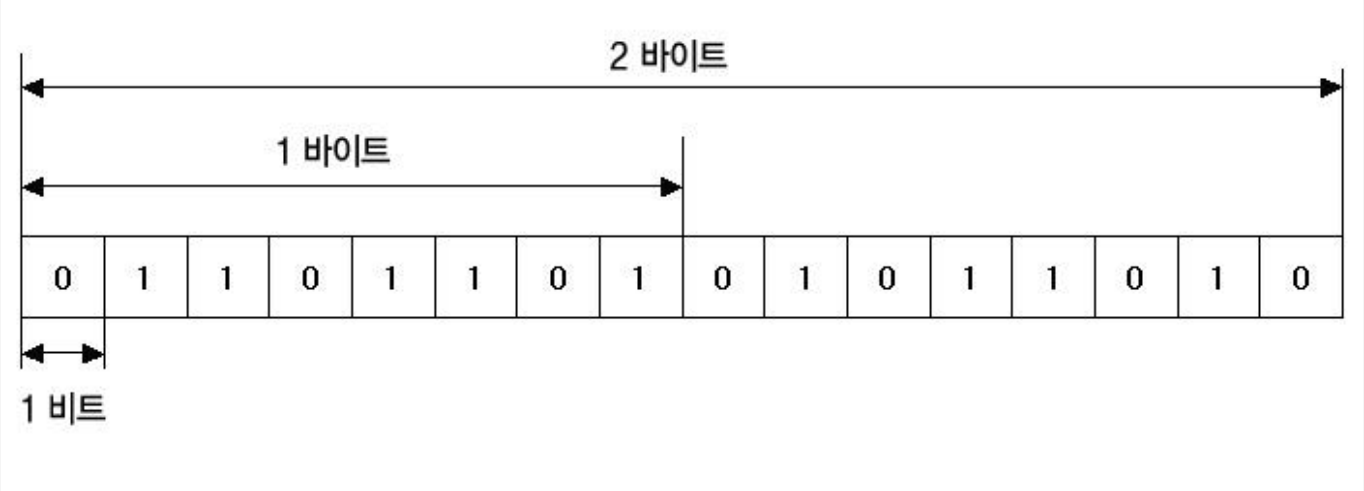
- 2진수와 10진수
  - 10진수 : 0~9를 이용한 데이터의 표현
  - 2진수 : 0과 1을 이용한 데이터의 표현
  - 컴퓨터는 내부적으로 모든 데이터 2진수로 처리



- 16진수와 10진수
  - 16진수 : 0~9, a, b, c, d, e, f를 이용한 데이터의 표현



- 데이터의 표현 단위인 비트(bit)와 바이트(byte)
  - 비트 : 데이터 표현의 최소 단위, 2진수 값 하나 (0 or 1)을 저장
  - 바이트 : 8비트 == 1바이트



- 연습문제

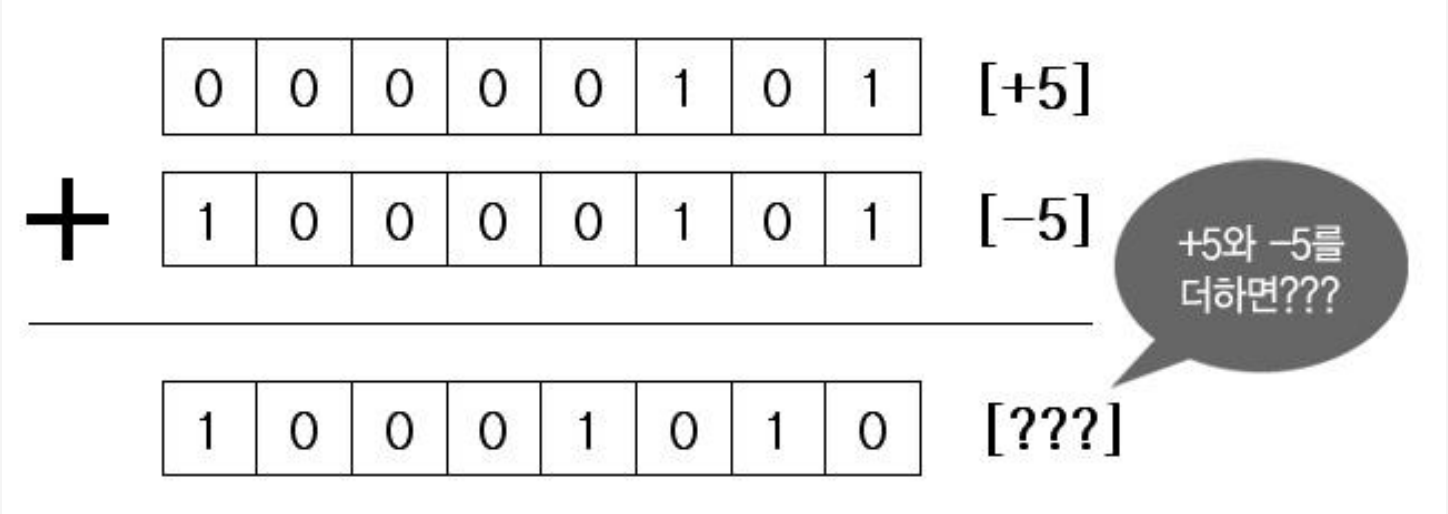
0	0	0	0	0	0	0	0	[ ]
0	0	0	0	0	0	0	1	[ ]
0	0	0	0	0	0	1	0	[ ]
0	0	0	0	0	1	0	0	[ ]
0	0	0	0	1	0	0	0	[ ]
0	0	0	1	0	0	0	0	[ ]
0	0	1	0	0	0	0	0	[ ]
0	1	0	0	0	0	0	0	[ ]
1	0	0	0	0	0	0	0	[ ]

- 프로그램상에서의 8진수, 16진수 표현
  - 8진수 : 0으로 시작
  - 16진수 : 0x로 시작

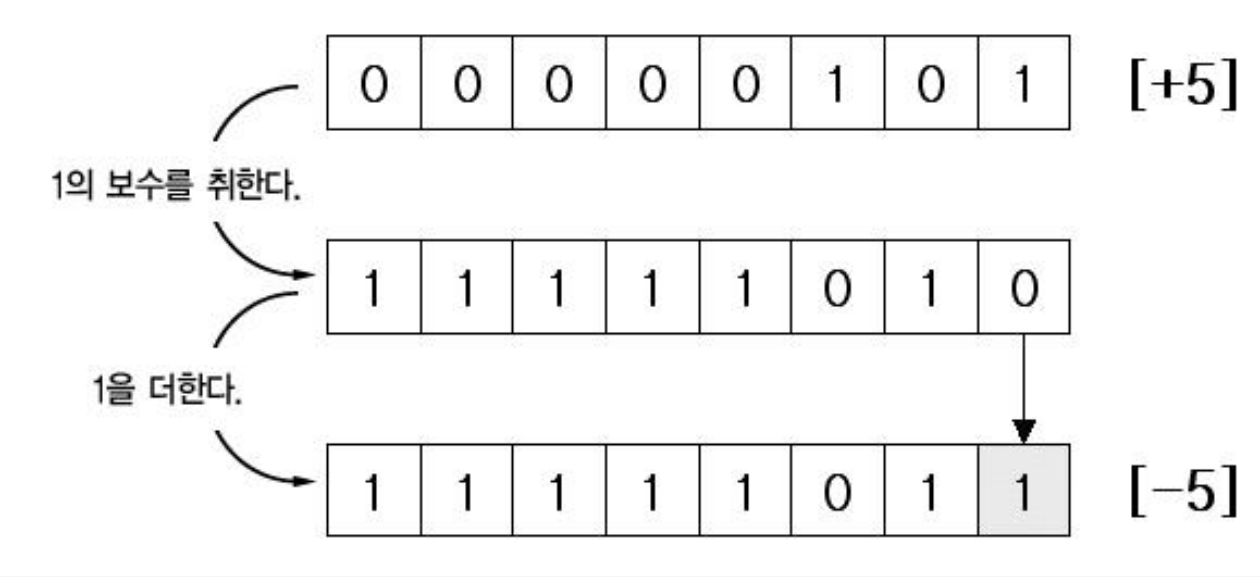
```
int a = 10;          // 10진수. 아무런 표시도 없으므로…
int b = 0xa;         // 16진수. 0x로 시작하므로…
int c = 012;         // 8진수. 0으로 시작하므로…
```

- 정수의 표현 방식
  - MSB : 가장 왼쪽 비트, 부호를 표현
  - MSB를 제외한 나머지 비트 : 데이터의 크기 표현

- 잘못된 음의 정수 표현 방식
  - 양의 정수 표현 방식을 적용한 경우



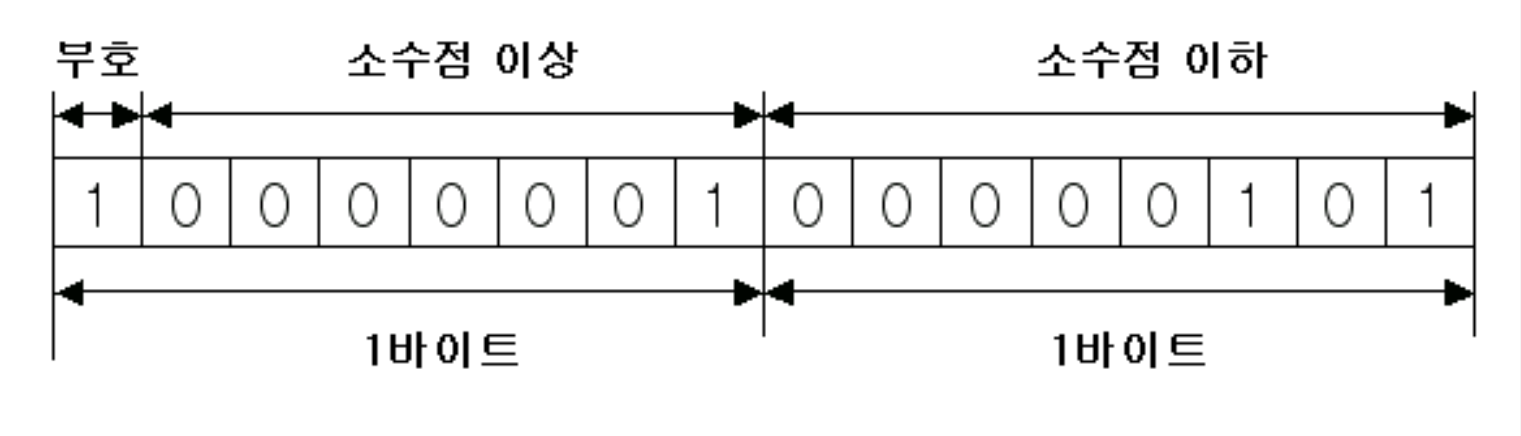
- 정확한 음의 정수 표현 방식
  - 2의 보수를 이용한 음의 정수 표현 방식



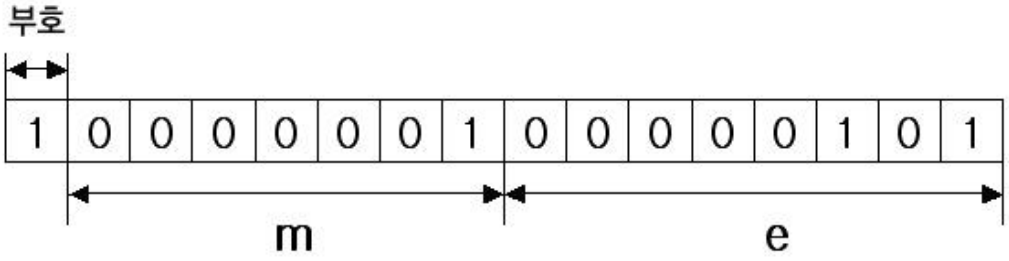
- 음수 표현 방식의 증명

- 연습문제
  - 양의 정수
  - 01001111 [ ]
  - 00110011 [ ]
  - 음의 정수
  - 10101001 [ ]
  - 11110000 [ ]

- 잘못된 실수의 표현 방식
  - 정수를 표현하는 방식을 실수 표현에 적용
  - 작은 수를 표현하는데 있어서 한계를 지님



- 정확한 실수 표현 방식
  - 오차가 존재하는 단점을 지님, 그러나 효율적인 표현 방식



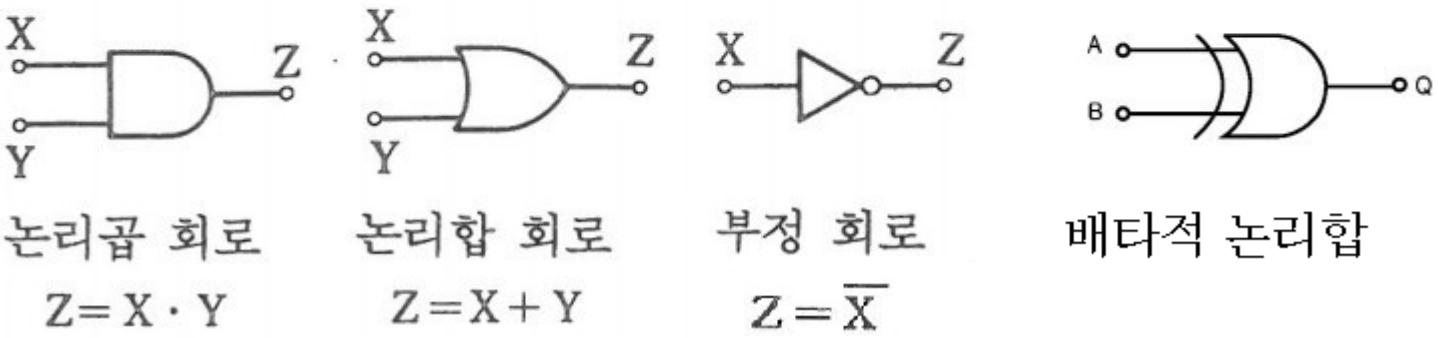
$$\pm(1.m)\times 2^{e-127}$$

- 비트 연산 : 컴퓨터의 내부 데이터 표현 방식인 비트(bit) 단위의 연산.

진리표

논리 기호

논리대수에 의한 연산을 하기 위한 연산자나 기능을 나타내는 기호로, 논리 회로를 도시하기 위해 사용한다. 각종 형식이 있으나 예를 들면 그림과 같은 것이 일반적으로 쓰인다



- 비트 단위 연산자의 종류

연산자	연산자의 의미	결합성
&	비트 단위 AND ex) a & b	→
|	비트 단위 OR ex) a | b	
^	비트 단위 XOR ex) a ^ b	
~	비트 단위 NOT ex) ~a	
<<	왼쪽으로 이동 ex) a << 2	
>>	오른쪽으로 이동 ex) a >> 2	

- & 연산자 : 비트 단위 AND

```
0 & 0  →  0을 반환
0 & 1  →  0을 반환
1 & 0  →  0을 반환
1 & 1  →  1을 반환
```

```
int main(void)
{
    int a=15;        // 00000000  00000000  00000000  00001111
    int b=20;        // 00000000  00000000  00000000  00010100
    int c = a&b;

    printf("AND 연산 결과 : %d", c);     // 출력 결과 4
```

- & 연산자 : 비트 단위 AND

```
        00000000 00000000 00000000 00001111
AND     00000000 00000000 00000000 00010100
-------------------------------------------
        00000000 00000000 00000000 00000100
```

그림 4-13

- | 연산자 : 비트 단위 OR

```
0 | 0  →  0을 반환
0 | 1  →  1을 반환
1 | 0  →  1을 반환
1 | 1  →  1을 반환
```

```
int main(void)
{
    int a=15;    // 00000000  00000000  00000000  00001111
    int b=20;    // 00000000  00000000  00000000  00010100
    int c = a|b;

    printf("OR 연산 결과 : %d", c);        // 출력 결과 31
```

4-3 비트 단위 연산

- | 연산자 : 비트 단위 OR

```
        00000000 00000000 00000000 00001111
OR      00000000 00000000 00000000 00010100
-------------------------------------------
        00000000 00000000 00000000 00011111
```

- ^ 연산자 : 비트 단위 XOR

```
0 ^ 0  →  0을 반환
0 ^ 1  →  1을 반환
1 ^ 0  →  1을 반환
1 ^ 1  →  0을 반환
```

```
int main(void)
{
    int a=15;   // 00000000  00000000  00000000  00001111
    int b=20;   // 00000000  00000000  00000000  00010100
    int c = a^b;

    printf("XOR 연산 결과 : %d", c);     // 출력 결과 27
```

- ^ 연산자 : 비트 단위 XOR

```
      00000000 00000000 00000000 00001111
XOR   00000000 00000000 00000000 00010100
-------------------------------------------
      00000000 00000000 00000000 00011011
```

- ~ 연산자 : 비트 단위 NOT

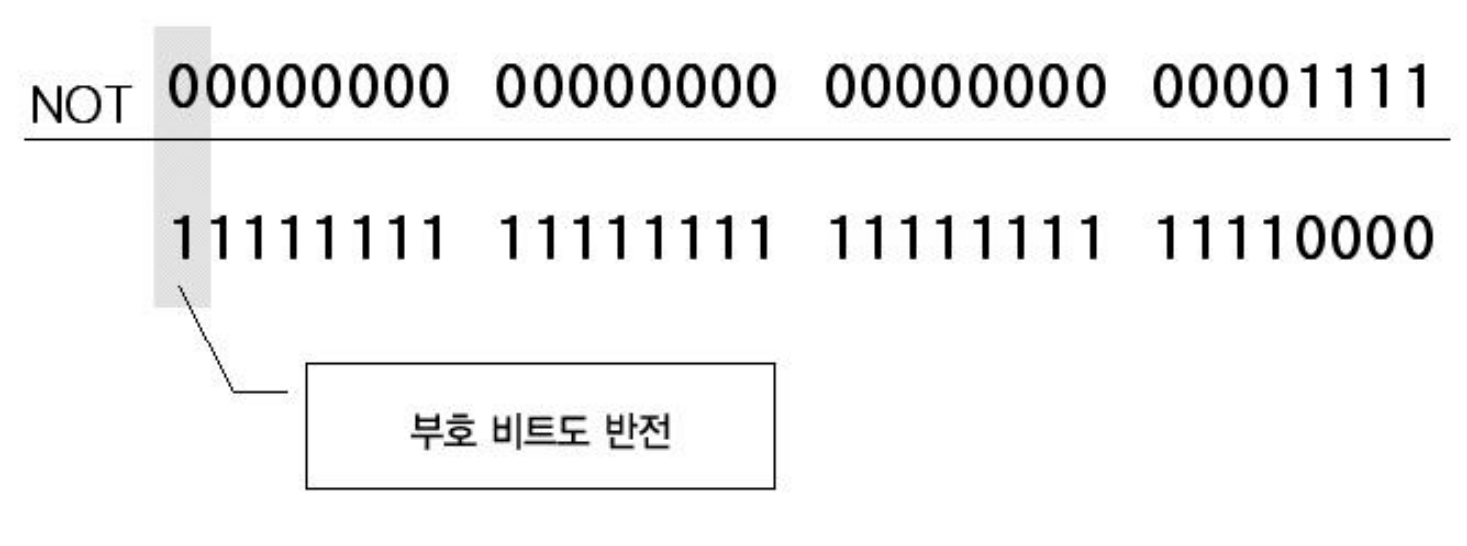
~ 0 → 1을 반환

~ 1 → 0을 반환

```
int main(void)
{
    int a=15;
    int b=~a;

    printf("NOT 연산 결과 : %d", b);      // 출력 결과 -16
```

- ~ 연산자 : 비트 단위 NOT


NOT 00000000 00000000 00000000 00001111
11111111 11111111 11111111 11110000
부호 비트도 반전

- << 연산자 : 왼쪽 쉬프트(shift) 연산

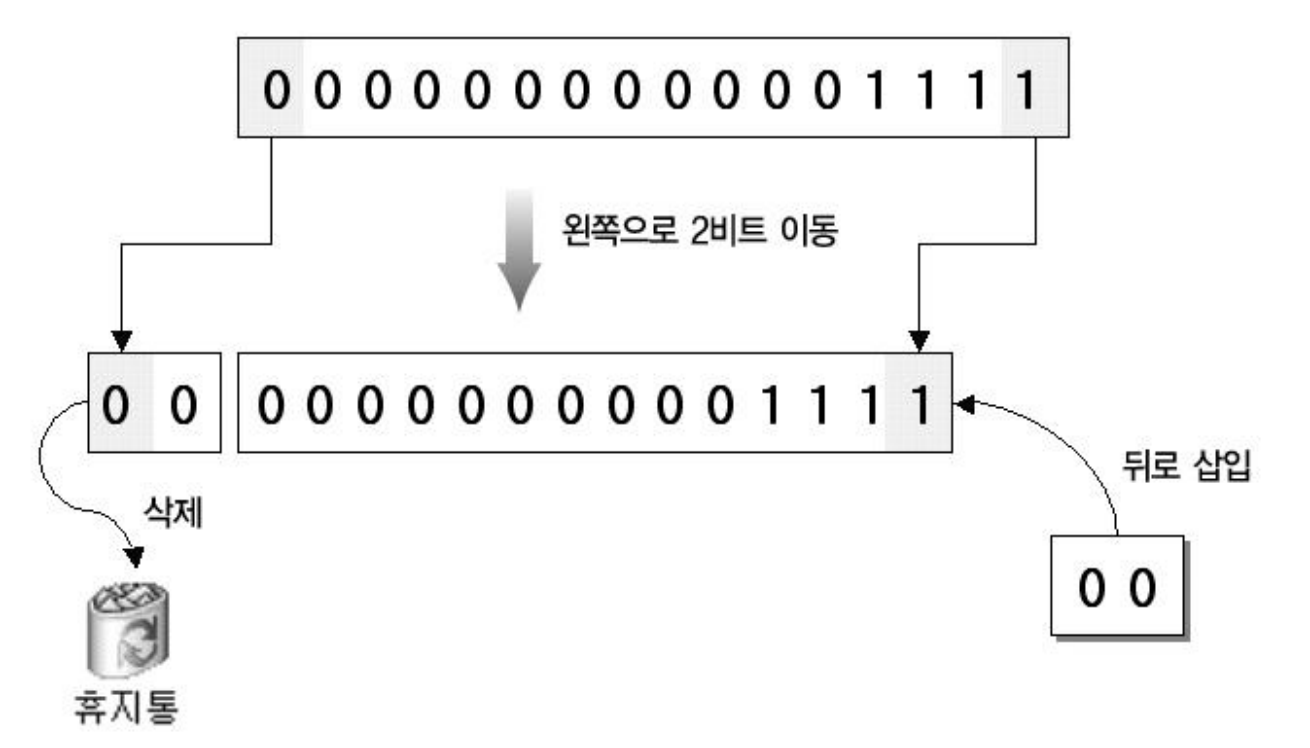
a<<b → a의 비트들을 b칸씩 왼쪽으로 이동한 값을 반환

8<<2 → 8의 비트들을 왼쪽으로 2칸씩 이동한 값을 반환

```
int main(void)
{
    int a=15;    // 00000000  00000000  00000000  00001111
    int b= a<<2;    // a의 비트들을 왼쪽으로 2칸씩 이동

    printf("<<2 연산 결과 : %d", b);      // 출력 결과 60
```

- << 연산자 : 왼쪽 쉬프트(shift) 연산



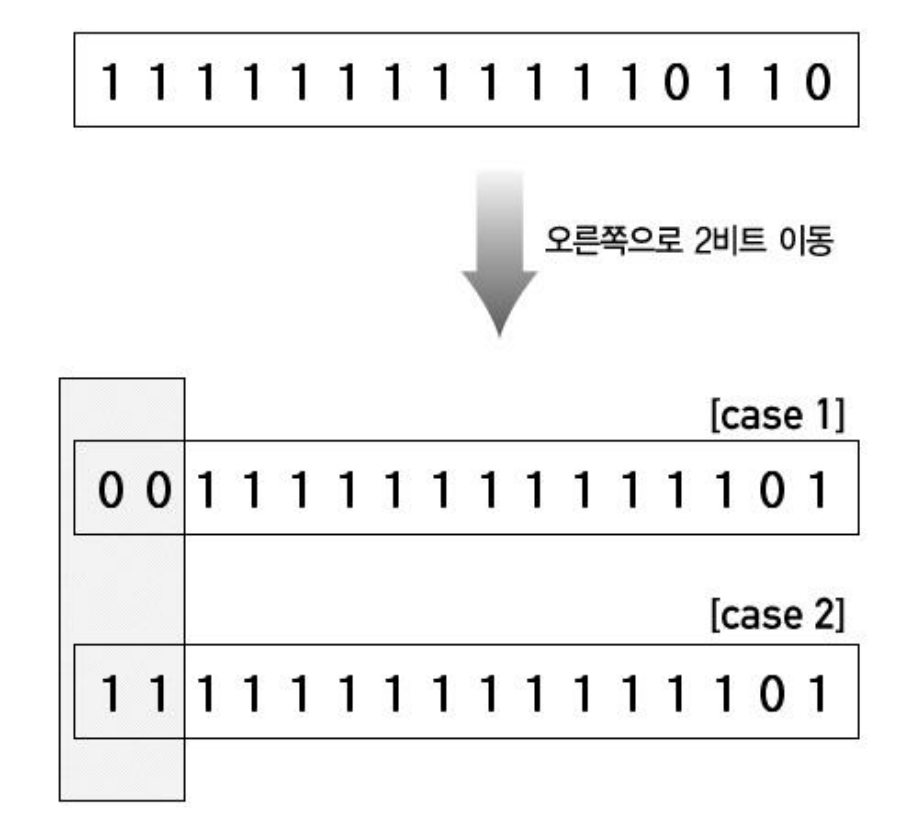
- >> 연산자 : 오른쪽 쉬프트(shift) 연산

a>>b → a의 비트들을 b칸씩 오른쪽으로 이동한 값을 반환

8>>2 → 8의 비트를 왼쪽으로 2칸씩 이동한 값을 반환

```
a=-10;
b=a>>2;   // a의 비트들을 2칸씩 오른쪽으로 이동한 값을 b에 저장
```

- >> 연산자 : 오른쪽 쉬프트(shift) 연산



5장. 상수와 기본 자료형

C 언어의 기본 자료형

- 자료형(data type)
  - "선언할 변수의 특징을 나타내기 위한 키워드"

```
Int  val;
```

- 기본 자료형
  - 기본적으로 제공이 되는 자료형

- 사용자 정의 자료형
  - 사용자가 정의하는 자료형 : 구조체, 공용체

- 기본 자료형 종류와 데이터의 표현 범위

자료형(data type)		할당되는 메모리 크기	표현 가능한 데이터의 범위
정수형	char	1 바이트	-128 ~ +127
	short	2 바이트	-32768 ~ +32767
	int	4 바이트	-2147483648 ~ +2147483647
	long	4 바이트	-2147483648 ~ +2147483647
실수형	float	4 바이트	$3.4*10^{-37}$ ~ $3.4*10^{+38}$
	double	8 바이트	$1.7*10^{-307}$ ~ $1.7*10^{+308}$
	long double	8 바이트 혹은 그 이상	차이를 많이 보임

- 다양한 자료형이 제공되는 이유
  - 데이터의 표현 방식이 다르기 때문
    - 정수형 데이터를 표현하는 방식
    - 실수형 데이터를 표현하는 방식
  - 메모리 공간을 적절히 사용하기 위해서
    - 데이터의 표현 범위를 고려해서 자료형 선택
    - 작은 메모리 공간에 큰 데이터를 저장하는 경우 데이터 손실이 발생할 수 있음

- sizeof 연산자
  - 피연산자의 메모리 크기를 반환
  - 피연산자로 자료형의 이름이 올 경우 괄호를 사용
  - 그 이외의 경우 괄호의 사용은 선택적

```
int main(void)
{
    int val=10;
    printf("%d",  sizeof val  );     // 변수 val의 메모리 크기 출력
    printf("%d",  sizeof(int) );     // 자료형 int의 메모리 크기 출력
     . . . . .
```

- 자료형 선택의 기준
  - 정수형 데이터를 처리하는 경우
    - 컴퓨터는 내부적으로 int형 연산을 가장 빠르게 처리,
      따라서 정수형 변수는 int형으로 선언
    - 범위가 int형 변수를 넘어가는 경우 long형으로 선언
    - 값의 범위가 -128 ~ +127 사이라 할지라도 int형으로 선언

- 자료형 선택의 기준
  - 실수형 데이터를 처리하는 경우
    - 선택의 지표는 정밀도
    - 정밀도란 오차 없이 표현 가능한 정도를 의미함
    - 일반적 선택은 double

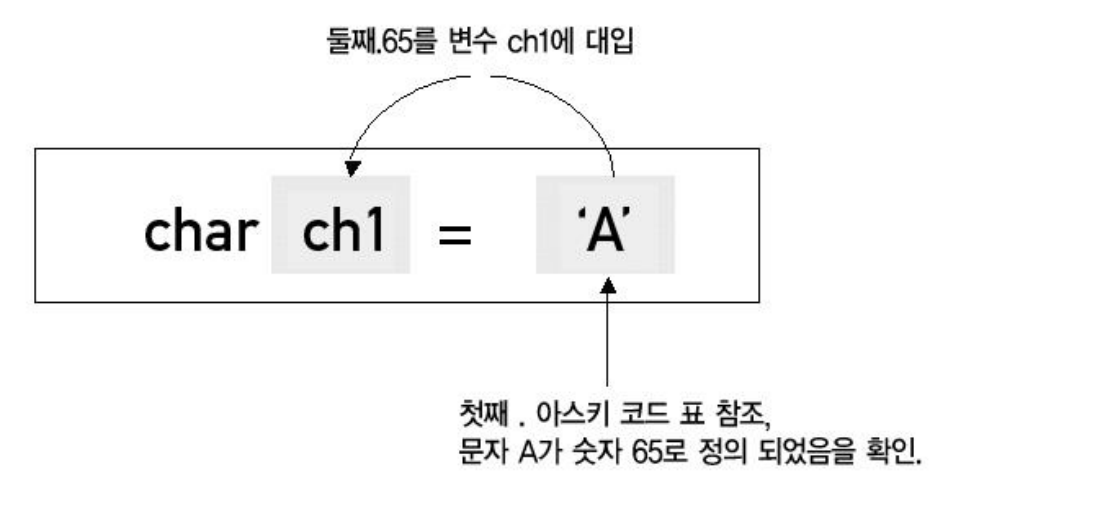
자료형	정밀도
float	소수 이하 6자리
double	소수 이하 15자리
long double	double의 정밀도와 같거나 크다.

- unsigned가 붙어서 달라지는 표현의 범위
  - MSB까지도 데이터의 크기를 표현하는데 사용
  - 양의 정수로 인식
  - 실수형 자료형에는 붙일 수 없다.

자료형	메모리 크기	표현 가능한 데이터의 범위
char(signed char)	1바이트	-128 ~ +127
unsigned char	1바이트	0 ~ (127 + 128)
short(signed short)	2바이트	-32768 ~ +32767
unsigned short	2바이트	0 ~ (32767 + 32768)
int(signed int)	4바이트	-2147483648 ~ +2147483647
unsigned int	4바이트	0 ~ (2147483647 + 2147483648)
long(signed long)	4바이트	-2147483648 ~ +2147483647
unsigned long	4바이트	0 ~ (2147483647 + 2147483648)

- 문자 표현을 위한 ASCII 코드의 등장
  - 미국 표준 협회(ANSI)에 의해 정의
  - 컴퓨터를 통해서 문자를 표현하기 위한 표준
    - 컴퓨터는 문자를 표현하지 못함
  - 문자와 숫자의 연결 관계를 정의
    - 문자 A는 숫자 65, 문자 B는 숫자 66…

- ASCII 코드의 범위
  - 0이상 127이하, char형 변수로 처리 가능
  - char형으로 처리하는 것이 합리적
- 문자의 표현
  - 따옴표(' ')를 이용해서 표현

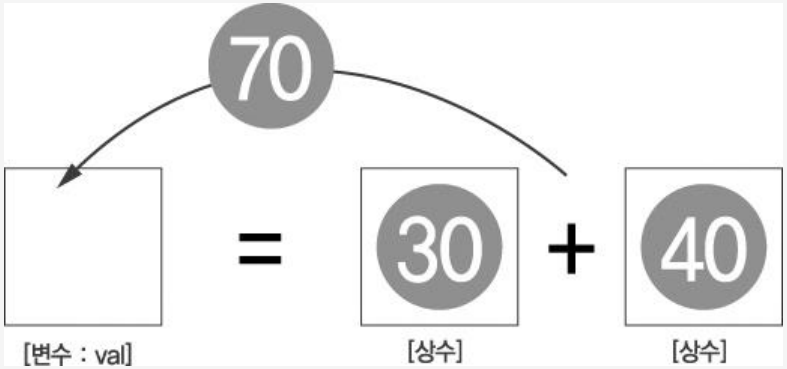


• ASCII 코드

Dec	Hx	Char		Dec	Hx	Chr	Dec	Hx	Chr	Dec	Hx	Chr
0	0	NUL	(null)	32	20	Space	64	40	@	96	60	`
1	1	SOH	(start of heading)	33	21	!	65	41	A	97	61	a
2	2	STX	(start of text)	34	22	"	66	42	B	98	62	b
3	3	ETX	(end of text)	35	23	#	67	43	C	99	63	c
4	4	EOT	(end of transmission)	36	24	$	68	44	D	100	64	d
5	5	ENQ	(enquiry)	37	25	%	69	45	E	101	65	e
6	6	ACK	(acknowledge)	38	26	&	70	46	F	102	66	f
7	7	BEL	(bell)	39	27	'	71	47	G	103	67	g
8	8	BS	(backspace)	40	28	(	72	48	H	104	68	h
9	9	TAB	(horizontal tab)	41	29	)	73	49	I	105	69	i
10	A	LF	(NL line feed, new line)	42	2A	*	74	4A	J	106	6A	j
11	B	VT	(vertical tab)	43	2B	+	75	4B	K	107	6B	k
12	C	FF	(NP form feed, new page)	44	2C	,	76	4C	L	108	6C	l
13	D	CR	(carriage return)	45	2D	-	77	4D	M	109	6D	m
14	E	SO	(shift out)	46	2E	.	78	4E	N	110	6E	n
15	F	SI	(shift in)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	(data link escape)	48	30	0	80	50	P	112	70	p
17	11	DC1	(device control 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2	(device control 2)	50	32	2	82	52	R	114	72	r
19	13	DC3	(device control 3)	51	33	3	83	53	S	115	73	s
20	14	DC4	(device control 4)	52	34	4	84	54	T	116	74	t
21	15	NAK	(negative acknowledge)	53	35	5	85	55	U	117	75	u
22	16	SYN	(synchronous idle)	54	36	6	86	56	V	118	76	v
23	17	ETB	(end of trans. block)	55	37	7	87	57	W	119	77	w
24	18	CAN	(cancel)	56	38	8	88	58	X	120	78	x
25	19	EM	(end of medium)	57	39	9	89	59	Y	121	79	y
26	1A	SUB	(substitute)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	(escape)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	(file separator)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	(group separator)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	(record separator)	62	3E	>	94	5E	^	126	7E	~
31	1F	US	(unit separator)	63	3F	?	95	5F	_	127	7F	DEL

- 리터럴(literal) 상수
  - 이름을 지니지 않는 상수

```
int main(void)
{
    int val = 30 + 40;
    . . . . .
```



- 리터럴 상수의 기본 자료형
  - 상수도 메모리 공간에 저장되기 위해서 자료형이 결정된다.

```
int main(void)
{
    char c = 'A';          // 문자상수(char)
    int i = 5;             // 정수상수(int)
    double d= 3.15;        // 실수상수(double)
    . . . . .
```

- 리터럴 상수의 기본 자료형

```
int main(void)
{
    float f = 3.14;     // float f= 3.14f
    return 0;
}
```

warning C4305: 'initializing' : truncation from 'const double ' to 'float '

- 접미사에 따른 다양한 상수의 표현

접미사	자료형	사용 예
u or U	unsigned int	304U
l or L	long	304L
ul or UL	unsigned long	304UL
f or F	float	3.15F
l or L	long double	3.15L

- 심볼릭(symbolic) 상수
  - 이름을 지니는 상수

- 심볼릭 상수를 정의하는 방법
  - const 키워드를 통한 변수의 상수화
  - 매크로를 이용한 상수의 정의

- const 키워드에 의한 상수화

```
int main(void)
{
    const  int MAX=100;
    const  double PI=3.1415;
    . . . . .
}
```

- 잘못된 상수 선언

```
int main(void)
{
    const  int MAX;
    MAX  = 100;
    . . . . .
}
```

- 자료형 변환의 두 가지 형태
  - 자동 형 변환
    - 자동적으로 발생하는 형태의 변환을 의미한다.
    - 묵시적 형 변환이라고도 표현한다.
  - 강제 형 변환
    - 프로그래머가 명시적으로 형 변환을 요청하는 형태의 변환
    - 명시적 형 변환이라고도 표현한다.

- 자동 형 변환이 발생하는 상황 1
  - 대입 연산 시

```
int main(void)
{
    int n=5.25;       // 소수부의 손실
    double d=3;       // 값의 표현이 넓은 범위로의 변환
    char c=129;       // 상위 비트의 손실
```

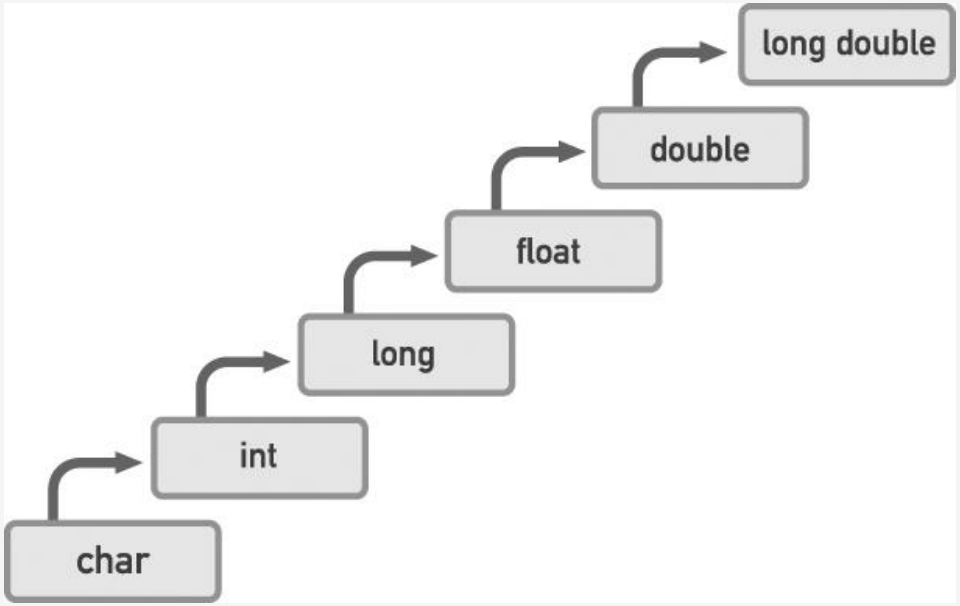
- 자동 형 변환이 발생하는 상황 2
  - 정수의 승격에 의해(int형 연산이 빠른 이유)
  - 정수형 연산 자체를 단일화시킨 결과

```
int main(void)
{
    char c1=10, c2=20;
    char c3=c1+c2;
    . . . . .
```

- 자동 형 변환이 발생하는 상황 3
  - 산술 연산 과정에 의해

```
int main(void)
{
    double e1 = 5.5 + 7;              // double + int
    double e2 = 3.14f+5.25; // float + double
    . . . . .
```

- 산술 연산 형 변환 규칙
  - 데이터의 손실이 최소화되는 방향으로...



- 강제 형 변환
  - 프로그래머의 요청에 의한 형 변환
  - Cast 연산자

```
float f= (float)3.14;          // 3.14를 float 형으로 형 변환
double e1 = 3 + 3.14;          //정수 30이 double 형으로 자동 형 변환
double e2 = 3 + (int)3.14;     // 3.14가 int형으로 강제 형 변환
```