

프로그래밍언어 실습

**실습 2 – 10진수 숫자 처리와 2진수 표현,  
조건문과 반복문 (보충설명)**



**교수 김영탁, 황현동**  
**영남대학교 정보통신공학과**  
(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

# Outline

- ◆ C 프로그램의 기본 자료 유형
- ◆ 컴퓨터 내부에서의 숫자의 표현
- ◆ `printf()`, `scanf()`에서의 자료 유형 및 포맷 지정
- ◆ 프로그램 실행 제어 기초 - 조건문, 반복문
- ◆ 비트단위 연산



## **C 프로그램에서의 숫자 표현 및 입출력 포맷 지정**

# 자료형의 종류

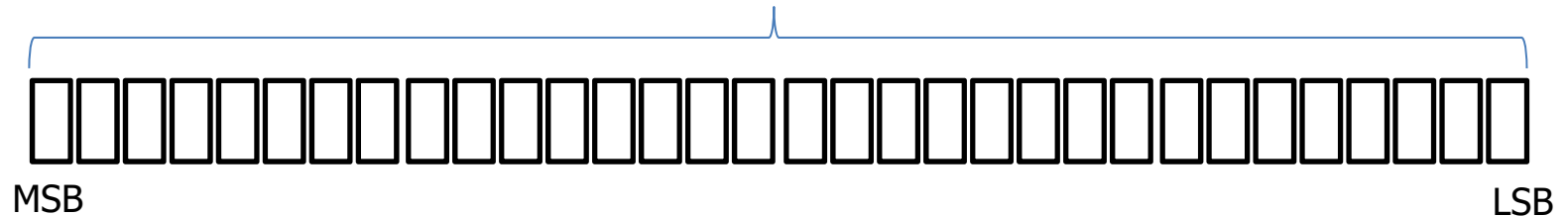
자료형			설명	바이트 수 (비트수)	범위
문자형	부호 있음	char	문자 및 정수	1 (8)	-128 ~ 127
	부호 없음	unsigned char	문자 및 부호없는 정수		0 ~ 255
정수형	부호 있음	short	short형 정수	2 (16)	-32768 ~ 32767
		int	정수	4 (32)	-2147483648 ~ 2147483647
		long	long형 정수		-2147483648 ~ 2147483647
	부호 없음	unsigned short	부호 없는 short형 정수	2 (16)	0 ~ 65535
		unsigned int	부호 없는 정수	4 (32)	0 ~ 4294967295
		unsigned long	부호 없는 long형 정수		0 ~ 4294967295
부동 소수점형		float	단일정밀도 부동소수점	4 (32)	1.2E-38 ~ 3.4E38
		double	두배 정밀도 부동소수점	8 (64)	2.2E-308 ~ 1.8E308
		Long double	두배 정밀도 부동소수점		2.2E-308 ~ 1.8E308



# 정수 (integer)의 비트 단위 표현

(a) unsigned integer

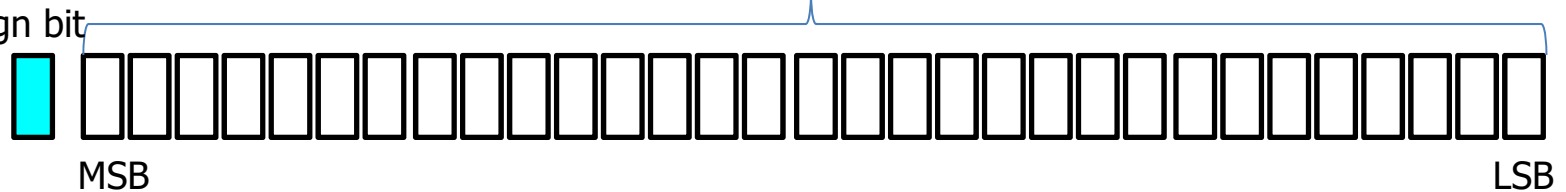
32-비트로 정수 표현



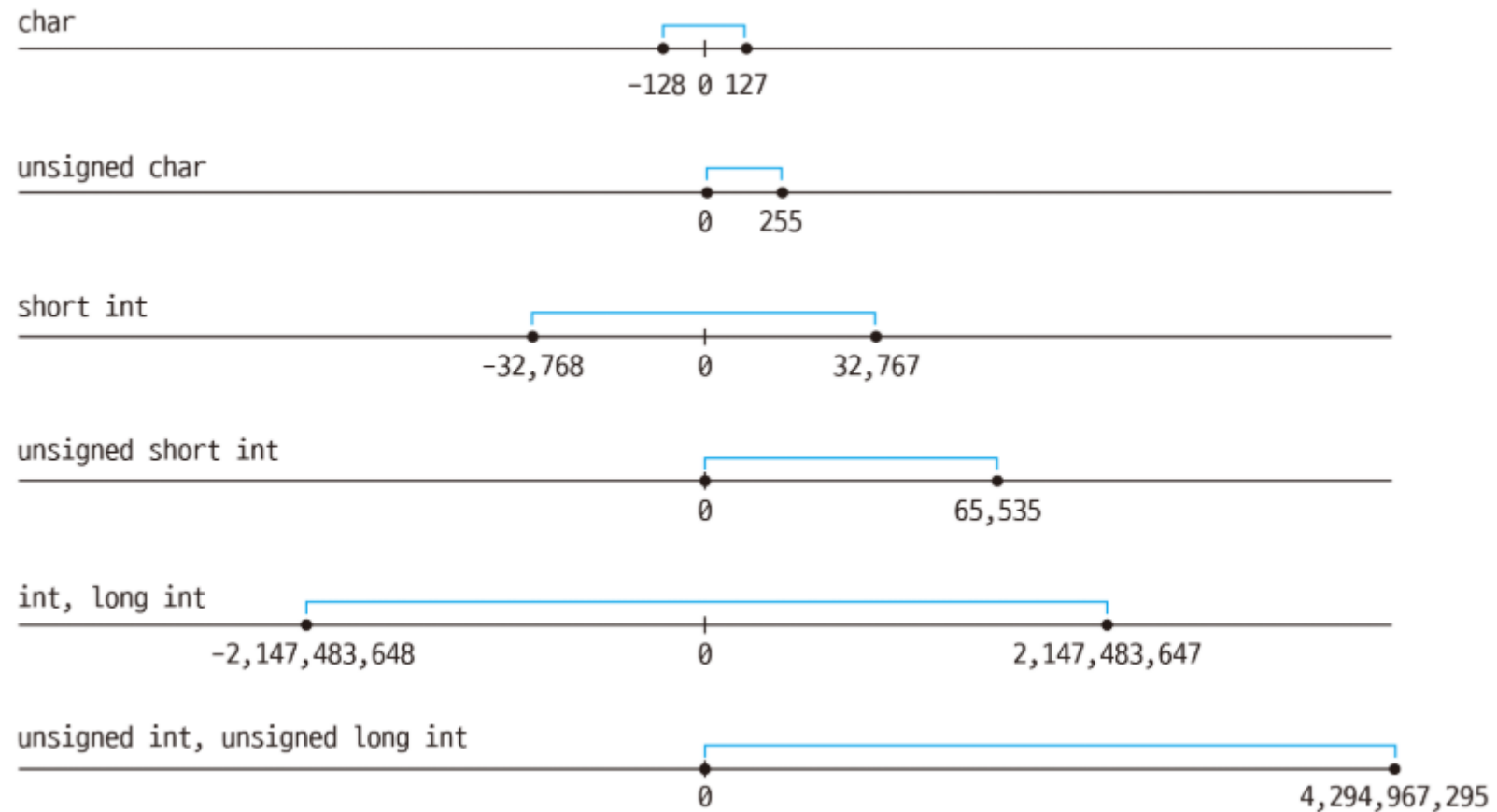
(b) signed integer

31-비트로 정수 표현

sign bit



# 정수형 (integer) 크기 및 범위



# 정수 (integer)의 2의 보수 (2's complement) 표현

## ◆ 8-bit 정수의 2의 보수 (2's complement) 표현

Decimal Value	Binary (2's complement representation)
127	0111 1111
126	0111 1110
2	0000 0010
1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
-126	1000 0010
-127	1000 0001
-128	1000 0000

정수 (integer)의 음수 (negative number)를 2의 보수 (2's complement)로 표현하는 방법:

- (1) 해당 정수의 비트 표현에서 0을 1로, 1을 0으로 변환 (1's complement)
- (2) 결과에 1을 더함

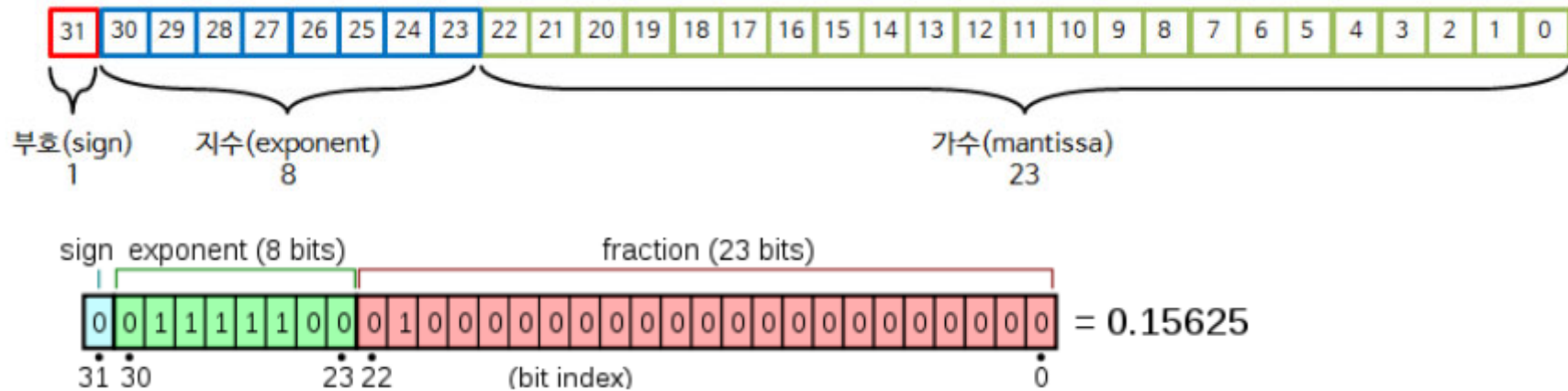
예) -127

(1) 127    (0111 1111) → (1000 0000)  
 (2) +1                 +(0000 0001)

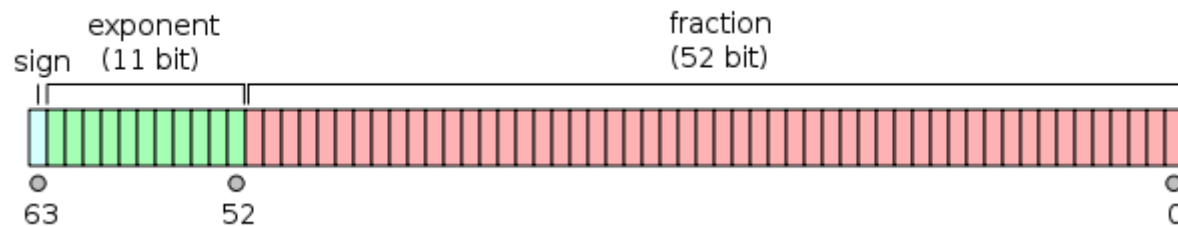
**1000 0001**

# 실수 (float)의 비트 단위 표현

## ◆ IEEE 754 float 표준 - 4 bytes (32 bit)



## ◆ IEEE 754 double 표준 - 8 bytes (64 bit)





# 숫자 (Number)의 표현

## ◆ 숫자의 표현

- 10진수 (decimal)
- 8진수 (octal)
- 16진수 (hexa-decimal)
- 2진수 (binary)

## ◆ 다양한 숫자 표현의 예

- (10진수) 52
- ( 8진수) 064
- (16진수) 0x34
- ( 2진수) 0011 0100



# 10진수의 자리수 별 값 산출

## ◆ 10진수의 자리수 별 값

- $345 = 3 \times 100 + 4 \times 10 + 5$   
 $= 3\text{백 } 4\text{십 } 5$

## ◆ 10진수의 각 자리수 별 값 산출

- 주어진 값:  $x$   
(예: 345)
- 1의 자릿수 값:  $x \% 10$   
(예:  $345 \% 10 \rightarrow 5$ )
- 10의 자릿수 값:  $(x / 10) \% 10$   
(예:  $345 / 10 \rightarrow 34$ ;  $34 \% 10 \rightarrow 4$ )
- 100의 자릿수 값:  $(x / 100) \% 10$   
(예:  $345 / 100 \rightarrow 3$ ;  $3 \% 10 \rightarrow 3$ )



# 주어진 값에 대한 2진수 값 산출

## ◆ 예)

- (10진수) 52
- (2진수) 0011 0100 =  $0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4$   
 $+ 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$   
 $= 32 + 16 + 4 = 52$

## ◆ 2진수 값의 산출

- 주어진 2진수 값: b
- $2^0$  자리 값의 산출:  $b \% 2$  또는  $b \& 00000001$
- $2^1$  자리 값의 산출:  $(b / 2) \% 2$  또는  $(b >> 1) \& 0x01$
- $2^2$  자리 값의 산출:  $(b >> 2) \& 0x01$



# ASCII (Character Set)

## ◆ ASCII (American Standard Code for Information ) Interchange

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL



# Formatted output in C - printf()

## ◆ Example

code: `printf( "Color %s, Number %d, Float %5.2f" , "red", 123456, 3.14);`

Output: **Color red, Number 123456, Float 3.14**

```
int x = 7;
double d = 12.5;
printf("%4d", x); // prints out integer x in 4 places
printf("%2d", 3); // prints out " 3"
printf("%02d", 3); // prints out "03".
printf("%4d, %4x", 15, 15); // prints out "15, F"
printf("%7.2f", d); // printout double d in total 7 places
                     // with 2 digits to the right of decimal points
```



## ◆ printf()에서의 포맷 지정자

Format Character	Output data type	Output
%d	int	signed decimal integer
%u	unsigned int	unsigned decimal integer
%o	unsigned int	unsigned octal integer
%x, %X	unsigned int	unsigned hexadecimal integer
%f	float, double	floating point numbers in decimal format
%e, %E	float, double	floating point numbers in scientific format (e.g., 1.2345e-001 or 1.0E-20)
%g, %G	float, double	selects %f or %e according to the value
%c	char	character
%s	char *	string indicated by a character pointer
%p	void *	address value of the pointer
%n	int *	address value of the pointer



## ◆ Additional format specifier

- `printf("%d, %o, %x\\n", data, data, data);`  
// decimal, octal, hexa-decimal
- `printf("%4d, %4o, %4x\\n", data, data, data);`  
// “%4d”: print in 4 spaces, right-alignment
- `printf("%#d, %#o, %#X\\n", data, data, data);`  
// ‘#’ defines prefix of “0” for octal number,  
// 0x for hexa-decimal number
- `printf("%#08d, %#08o, %#08X\\n", data, data, data);`  
// “#0” defines filling leading zeros (‘0’) in front of the number
- `printf("%8d, %-8d, %+8d\\n", data, data, data);`  
// by default numbers are printed in right-alignment  
// ‘-’ defines left-alignment in printing numbers



## ◆ Examples

```
/** SimpleProg.cpp
 * Sample formatted output using printf()
 */
```

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

```
using namespace std;
```

```
void main()
```

```
{
```

```
    int data = 15;
```

```
    printf("%d, %o, %x\n", data, data, data);
```

```
        // decimal, octal, hexa-decimal
```

```
    printf("%4d, %4o, %4x\n", data, data, data);
```

```
        // 4 spaces of decimal, octal, hexa-decimal
```

```
    printf("%#d, %#o, %#X\n", data, data, data);
```

```
        // '#' defines prefix of "0" for octal number, 0xF for hexa-decimal number
```

```
    printf("%#08d, %#08o, %#08X\n", data, data, data);
```

```
        // filling leading zeros ('0') in spaces of the front of the number
```

```
    printf("%8d, %-8d, %+8d\n", data, data, data);
```

```
        // '-' defines left-alignment in prints of numbers
```

```
}
```

```
15, 17, f
   15,   17,    f
15, 017, 0XF
00000015, 00000017, 0X00000F
   15, 15      ,      +15
계속하려면 아무 키나 누르십시오 . . .
```

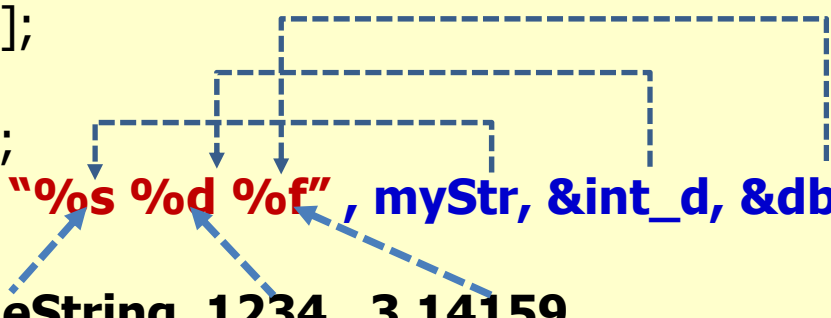




# Formatted input in C – scanf()

## ◆ Example

```
char myStr[20];  
int int_d;  
double dbl_d;  
code: scanf( "%s %d %f" , myStr, &int_d, &dbl_d);  
input: SampleString 1234 3.14159
```



## ◆ scanf()에서의 포맷 지정자

Format Character	Output data type	Output
%d	int	signed decimal integer
%i	int	signed decimal integer
%u	unsigned int	unsigned decimal integer
%o	unsigned int	unsigned octal integer
%x	unsigned int	unsigned hexadecimal integer
%c	char	character
%s	char *	string indicated by a character pointer
%p	void *	address value of the pointer
%f, %e, %g	float	signed floating point number



## **프로그램 실행 제어 – 조건문, 반복문**

# 조건문 (condition) 과 반복문 (loop)

## ◆ 조건문(if ~ else)

- **if** (data > 10)
- **if** (grade > 90)  
    { . . . . }
- else**  
    { . . . . }
- nested if  
    **if** (grade > 90)  
    {  
        **if** (total\_credit > 15)  
        {  
            printf("You can apply scholarship.");  
        }  
        **else**  
        {  
            printf("Your grade is excellent,  
                but your credit is not enough for scholarship !");  
        }  
    }



# 조건문 (condition) 과 반복문 (loop)

## ◆ for - loop

```
sum = 0;
for (int i=0; i<10; i++)
{
    sum = sum + i;
}
```

## ◆ while – loop

```
sum = 0;
int i = 0;
while (i < 10)
{
    sum = sum + i;
    i++;
}
```



# 조건문(if ~ else) 이란?

## ◆ if문의 용도

- 프로그램의 구동상 조건에 따라 결정을 내리고 선택적 실행을 위한 구문

## ◆ if문의 용법

```
if(조건식)
{
    statement;//실행할 내용
}
```



# if 문의 예시

```
/* Simple Program with if-statement */
```

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

```
int main(void)
```

```
{
```

```
    int number;
```

```
    printf("input a decimal number : ");
```

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

```
    if (number > 0) 조건식
```

```
        printf("input number(%d) is positive.\n", number);
```

```
    else if (number == 0)
```

```
        printf("input number(%d) is zero.\n", number);
```

```
    else
```

```
        printf("input number(%d) is negative.\n", number);
```

```
    return 0;
```

```
}
```

실행내용

```
input a decimal number : -1234
input number(-1234) is negative.
```

```
C:\MyC_Cpp_Progs\2020-2 Book (C-Prog Visual Studio 2019)\ch 2\
e(프로세스 31048개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```



# 조건식의 종류

## ◆ 조건 식의 종류에는 여러 가지 연산자가 존재!

- $>$ ,  $<$  : 크거나 작거나 기호
- $==$  : 같다는 equal 기호
- $\&\&$  : AND 기호 ( $A \&\& B$  : A와 B 모두)
- $||$  : OR 기호 ( $A||B$  : A 또는 B)
- $!$  : NOT 기호(  $!A$  : A가 아니면)

## ◆ 조건식은 동시에 여러 개가 조합 될 수 있다.

- eg)
  - $>=$ (크거나 같거나),
  - $<=$ (작거나 같거나),
  - $!=$ (같지 않으면)





# if문과 else문

```
/* Checking input character */
```

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

```
void main(void)
```

```
{
```

```
    char ch;
```

```
    printf("Input a character or digit : ");
```

```
    scanf("%c", &ch);
```

```
    if (((ch >= 'A') && (ch <= 'Z')) || ((ch >= 'a') && (ch <= 'z')))
```

```
        printf("input character (%c) is a letter. \n", ch);
```

```
    else
```

```
        printf("input character (%c) is not a letter. \n", ch);
```

```
}
```

if와 else의 쌍으로 존재 하며,  
if문이 만족하면 if문의 내용이 실행 되고,  
그게 아닐 시 else문의 내용이 실행된다.



## 중복 if문 (Nested if-statement)

- ◆ 자신의 프로그램에서 여러 조건을 동시에 만족시키기를 원할 때 다중 if 문으로 코딩 할 수 있다.

- ex) A, B, C 세 사람이 있을 때, 3명 모두 돈이 없다.  
이것을 컴퓨터 프로그래밍 코드로 이해를 하면.....

```
if(A != HAVE_MONEY)
{
    if (B != HAVE_MONEY)
    {
        if(C != HAVE_MONEY)
        {
            //따라서, 3명 모두 다 돈이 없다
        }
    }
}
```



# 다중 if문

```
/* Checking input character */
```

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

```
void main(void)
```

```
{
```

```
    char ch;
```

```
    printf("Input a character or digit : ");
```

```
    scanf("%c", &ch);
```

```
    if ((ch >= 'A') && (ch <= 'Z'))
```

```
        printf("input character (%c) is capital(upper case) letter. \n", ch);
```

```
    else if ((ch >= 'a') && (ch <= 'z'))
```

```
        printf("input character (%c) is lower case letter. \n", ch);
```

```
    else if ((ch >= '0') && (ch <= '9'))
```

```
        printf("input character (%c) is a digit. \n", ch);
```

```
    else
```

```
        printf("input character (%c) is not an alphabet nor a digit. \n", ch);
```

```
}
```

**// 여러 개의 조건을 순차적으로  
// 검사하고 싶을 때 사용한다.**



## 반복문 이란?

### ◆ 우리가 수학적 계산을 위한 코딩을 한다고 가정을 해보자!

- 만약 1~N까지의 모든 수를 더하기를 할 때, C 프로그래밍에서는 어떻게 해야 할까?
  - 1~N까지 변수를 모두 선언한다? 그리고 모두 더한다??

Smart하게 **반복문**을 이용한다!!



## 반복문의 예시 for문

- ◆ **for문**이란 어떤 조건에 참이 될 때 까지 수행하도록 하는 반복문이다.

```
for(초기식; 조건식; 증감식)
{
    //내용
}
```

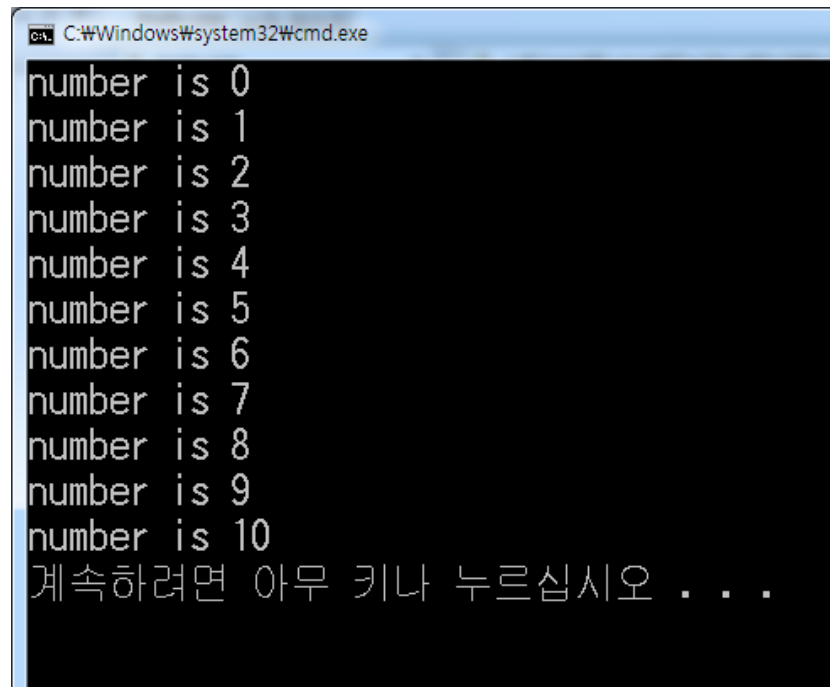


# for문의 예

```
#include <stdio.h>

int main(void)
{
    int number;
    for(number = 0; number <=10 ; number ++ )
    {
        printf("number is %d\n",number);
    }

    return 0;
}
```



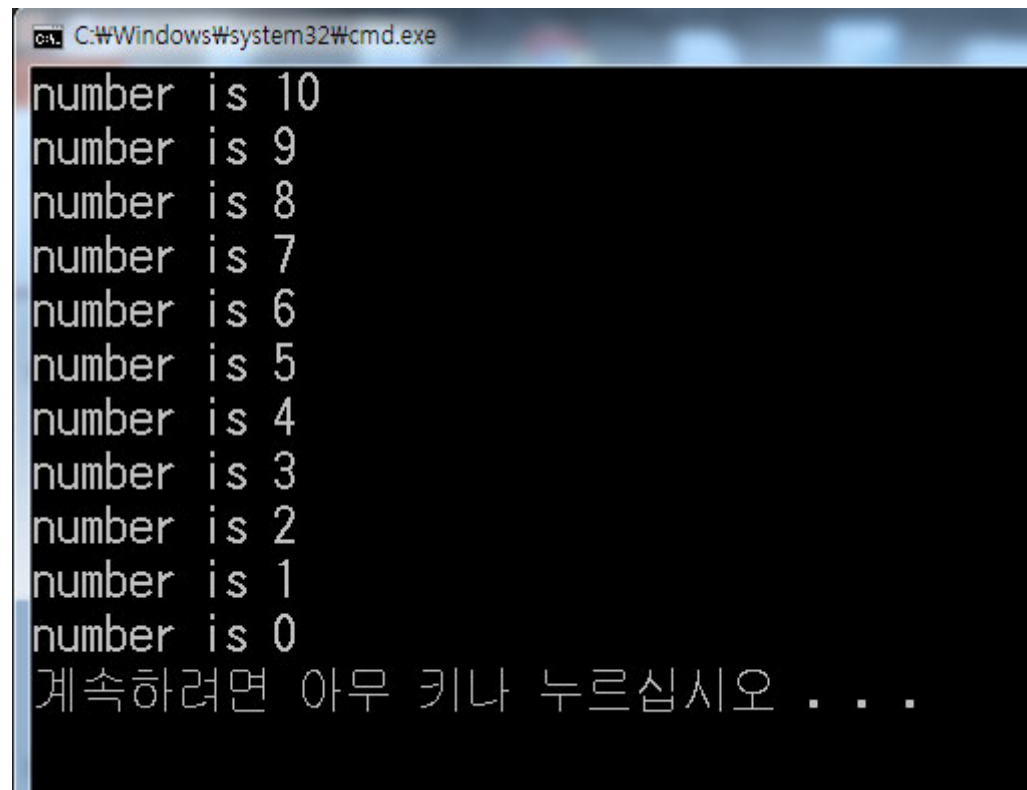
```
C:\Windows\system32\cmd.exe
number is 0
number is 1
number is 2
number is 3
number is 4
number is 5
number is 6
number is 7
number is 8
number is 9
number is 10
계속하려면 아무 키나 누르십시오 . . .
```



# for문의 예

```
#include <stdio.h>
-
int main(void)
{
    int number;
    for(number = 10; number >= 0 ; number --)
    {
        printf("number is %d\n", number);
    }

    return 0;
- }
```



```
C:\Windows\system32\cmd.exe
number is 10
number is 9
number is 8
number is 7
number is 6
number is 5
number is 4
number is 3
number is 2
number is 1
number is 0
계속하려면 아무 키나 누르십시오 . . .
```



# for문의 예

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

```
-
```

```
int main(void)
```

```
{
```

```
    int number;
```

```
    for(number = 0; number <=10 ; number = number+2)
```

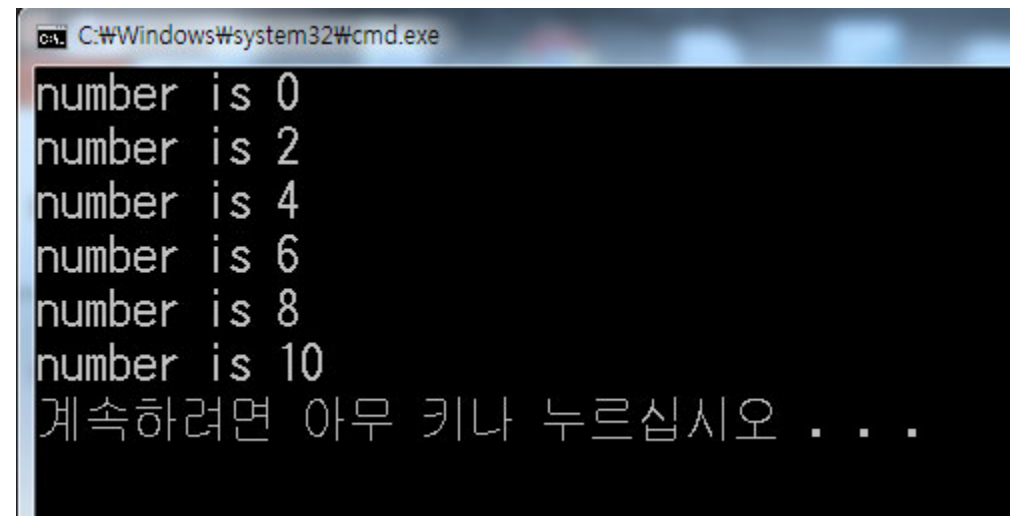
```
{
```

```
        printf("number is %d\n",number);
```

```
}
```

```
    return 0;
```

```
- }|
```



```
C:\Windows\system32\cmd.exe
number is 0
number is 2
number is 4
number is 6
number is 8
number is 10
계속하려면 아무 키나 누르십시오 . . .
```





# 다중 for 문 (nested for-loop)

## ◆ 1~9단 구구단을 계산하는 프로그램을 작성해보자

```
#include <stdio.h>

int main()
{
    int number1, number2;

    for (number1 = 1; number1 <= 9; number1=number1++)
    {
        printf("%d단\n", number1);
        for (number2 = 1; number2 <= 9; number2++)
        {
            printf("%2d x %2d = %2d\n", number1, number2, number1*number2);
        }
    }

    return 0;
}
```

```
C:\Windows\system32\cmd.exe
1단
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
2단
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
자리맞춤
```



# for문으로 별 그림 그리기

## ◆ 별로 4각형 그리기

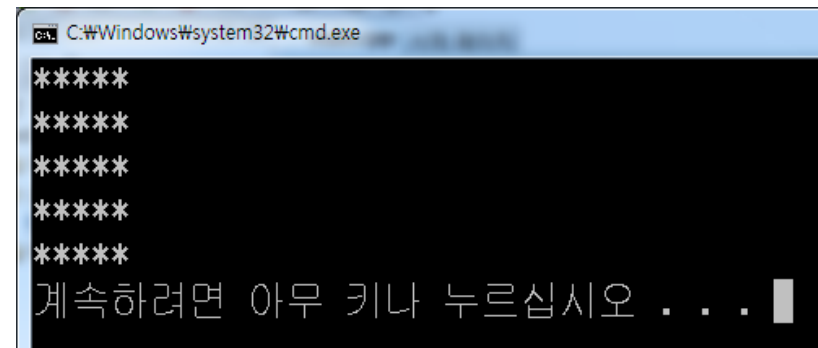
```
#include <stdio.h>

#define WIDTH 5
#define HEIGHT 5

int main()
{
    int i=0, j=0;

    for (i = 0; i < HEIGHT; i++)
    {
        for (j = 0; j < WIDTH; j++)
        {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
*****
*****
*****
*****
*****
계속하려면 아무 키나 누르십시오 . . .
```



# for문으로 별 그림 그리기

## ◆ 별로 중앙이 비어있는 사각형 그리기

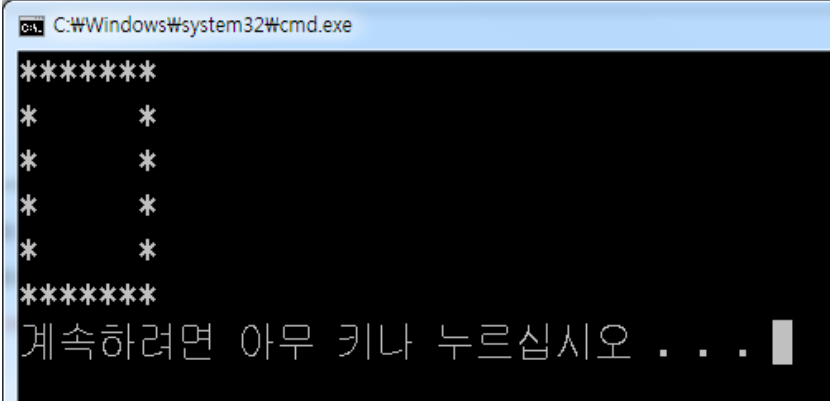
```
#include <stdio.h>

#define WIDTH 6
#define HEIGHT 5

int main()
{
    int i=0, j=0;

    for (i = 0; i <= HEIGHT; i++)
    {
        for (j = 0; j <= WIDTH; j++)
        {
            if (i == 0 || j == 0 || i == 5 || j == 6)
            {
                printf("*");
            }
            else
            {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
*****
*      *
*      *
*      *
*      *
*****
계속하려면 아무 키나 누르십시오 . . .
```

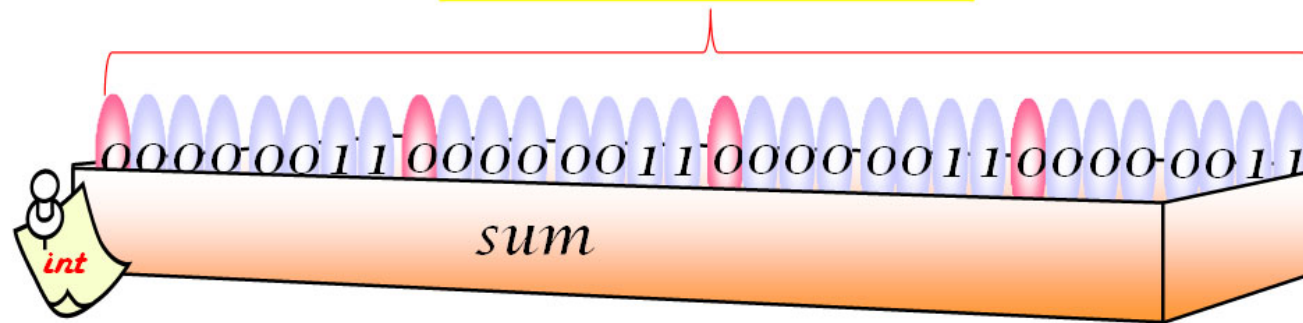


## 비트단위 연산

# 비트 연산자 (bit operator)

연산자 (Operator)	연산자의 의미	설명
&	비트 AND	두 피연산자의 해당 비트가 모두 1이면 1, 아니면 0
	비트 OR	두 피연산자의 해당 비트 중 하나만 1이면 1, 아니면 0
^	비트 XOR	두 피연산자의 해당 비트의 값이 같으면 0, 아니면 1
<<	왼쪽으로 이동	지정된 개수만큼 모든 비트를 왼쪽으로 이동한다.
>>	오른쪽으로 이동	지정된 개수만큼 모든 비트를 오른쪽으로 이동한다.
~	비트 NOT	0은 1로 만들고 1은 0로 만든다.

int 변수는 32비트로 되어 있다.



## 비트 AND 연산자 ('&')

0 AND 0 = 0
1 AND 0 = 0
0 AND 1 = 0
1 AND 1 = 1

변수1 00000000 00000000 00000000 00001001 (9)  
변수2 00000000 00000000 00000000 00001010 (10)

---

(변수1 AND 변수2) 00000000 00000000 00000000 00001000 (8)



# 비트 OR 연산자 ('|')

0 OR 0 = 0
1 OR 0 = 1
0 OR 1 = 1
1 OR 1 = 1

변수1	00000000	00000000	00000000	00001001	(9)
변수2	00000000	00000000	00000000	00001010	(10)
<hr/>					
(변수1 OR 변수2)	00000000	00000000	00000000	00001011	(11)



# 비트 XOR 연산자 ('^')

0 XOR 0 = 0
1 XOR 0 = 1
0 XOR 1 = 1
1 XOR 1 = 0

변수1 00000000 00000000 00000000 00001001 (9)  
변수2 00000000 00000000 00000000 00001010 (10)

---

(변수1 XOR 변수2) 00000000 00000000 00000000 00000011 (3)





# 비트 NOT 연산자 ('~')

NOT 0 = 1
NOT 1 = 0

부호비트가 반전되었기 때문에 음수가 된다.

변수1 00000000 00000000 00000000 00001001 (9)

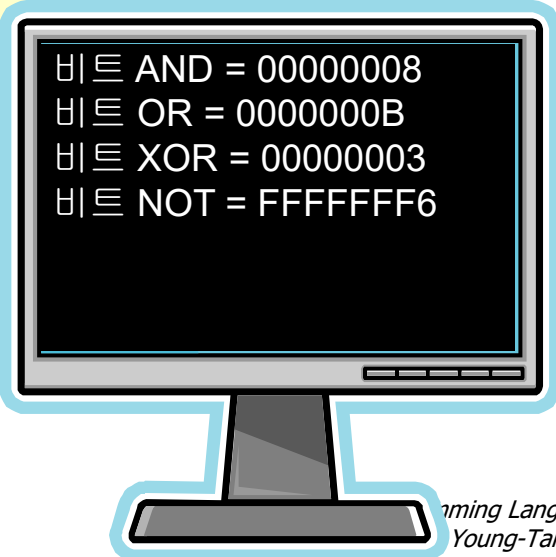
(NOT 변수1) 11111111 11111111 11111111 11110110 (-10)



## 예제: 비트 연산자

```
#include <stdio.h>
int main(void)
{
    int x = 9;           // 00001001
    int y = 10;          // 00001010
    printf("비트 AND = %08X", x & y); // 00001000
    printf("비트 OR = %08X", x | y);  // 00001011
    printf("비트 XOR = %08X", x ^ y); // 00000011
    printf("비트 NOT = %08X", ~x );   // 11110110

    return 0;
}
```

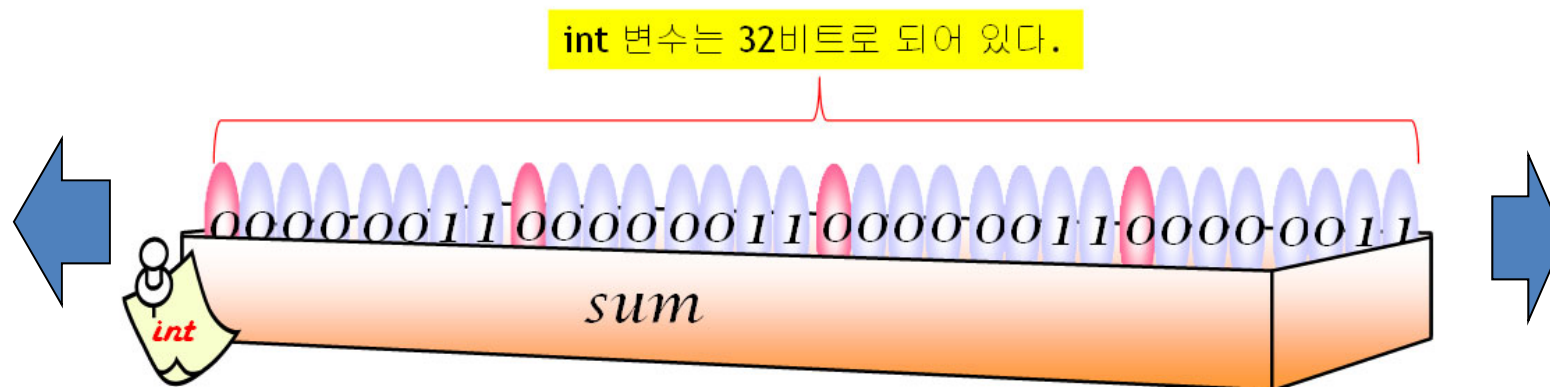


비트 AND = 00000008  
비트 OR = 0000000B  
비트 XOR = 00000003  
비트 NOT = FFFFFFFF6



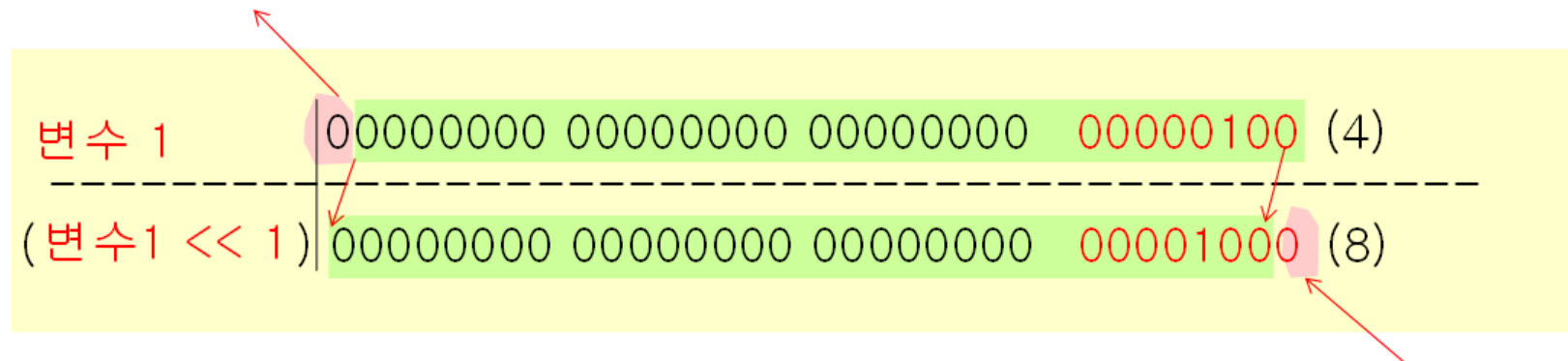
# 비트 이동 (bit shift) 연산자

연산자	기호	설명
왼쪽 비트 이동	<<	$x \ll y$ x의 비트들을 y 칸만큼 왼쪽으로 이동
오른쪽 비트 이동	>>	$x \gg y$ x의 비트들을 y 칸만큼 오른쪽으로 이동



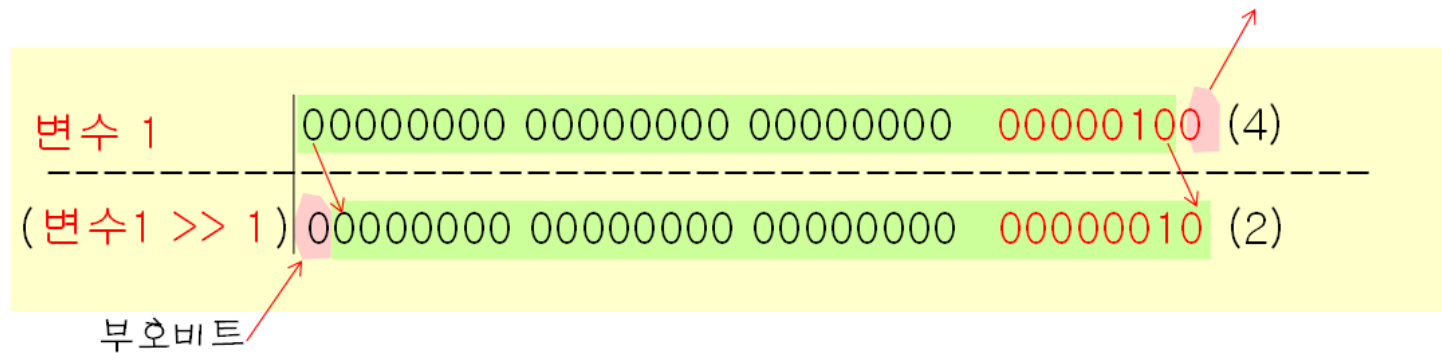
## << 연산자

- ◆ 비트를 왼쪽으로 이동
- ◆ 값은 2배가 된다.



## >> 연산자

- ◆ 비트를 오른쪽으로 이동
- ◆ 값은 1/2배가 된다.



## 예제: 비트 이동 연산자

```
#include <stdio.h>

int main(void)
{
    int x = 4;           // 0100

    printf("비트 << = %#08x", x << 1);    // 1000
    printf("비트 >> = %#08x", x >> 1);    // 0010

    return 0;
}
```



```
printInt_Bits(int d)
```

## printInt\_Bits(int d)

### ◆ 32-비트 (int) 크기의 정수 데이터 값을 비트 단위로 출력

- 정수형 데이터의 각 비트를 구분하여 얻는 방법으로 BIT\_MASK (0x01)의 bit-wise shift left 기능, bit-wise AND 기능, bit-wise shift right 기능으로 구현
- 방법: int 자료형에 포함된 bit 31 ~ bit 0을 bit 단위로 출력

```
01010010 00110011 11001100 10101010
& 10000000 00000000 00000000 00000000
-----
0
```





# printInt\_Bits(int d)

```
#define BIT_MASK 0x01
#define NUM_BITS_INT 32
void printInt_Bits(int d)
{
    unsigned long L1, L2, msk; // 32-bits
    for (int n=(NUM_BITS_INT - 1); n >= 0; n--)
    {
        msk = BIT_MASK << n;
        L1 = d & msk;
        L2 = L1 >> n;
        printf("%d", L2);
        if ((n % 8) == 0) // insert a space at each 8-bits
            printf(" ");
    }
}
```

n	msk (32-bits)	L1 (32-bits)	L2 (32-bits)	(decimal) L2
31				
30				
...				
1				
0				



# 정수형 데이터의 산술 연산 내용의 비트 단위 출력

## ◆ 정수형 연산자 종류를 **enum**으로 선언

```
enum BinOp_Type { INT_ADD, INT_SUB, INT_MUL, INT_DIV, INT_MOD, BIT_AND,  
                  BIT_OR, BIT_XOR, BIT_NOT, SHIFT_LEFT, SHIFT_RIGHT };
```

## ◆ 정수형 연산자의 표시를 위한 문자열 배열 선언

```
const char *bop_sign[] = { "+", "-", "*", "/", "%", "&", "|", "^", "~", "<<", ">>" };
```



# 정수형 데이터의 산술 연산 내용의 비트 단위 출력

- ◆ 3개의 정수형 (int) 데이터 (x, y, z)와 연산자 종류 (BinOp\_Type)를 전달받아 출력

```
void printIntOper_Bits(BinOp_Type bop, int x,  
int y, int z)
```

```
00000000 00000000 00000000 01010010  
+ 00000000 00000000 00000010 01000000  
-----  
00000000 00000000 00000010 10010010
```



# printIntOper\_Bits()

```
/* Definitions of BinOp_Types */

enum BinOp_Type {INT_ADD, INT_SUB, INT_MUL, INT_DIV, INT_MOD,
  BIT_AND, BIT_OR, BIT_XOR, BIT_NOT, SHIFT_LEFT, SHIFT_RIGHT };
const char *bop_sign[] = { "+", "-", "*", "/", "%", "&", "|", "^", "~", "<<",
  ">>" };

void printIntOper_Bits(BinOp_Type bop, int x, int y, int z)
{
    if (bop != BIT_NOT)
    {
        printf(" "); printInt_Bits(x); printf("\n");
    }
    printf(" %2s ", bop_sign[bop]); printInt_Bits(y); printf("\n");
    printf(" -----\n");
    printf(" "); printInt_Bits(z); printf("\n");
}
```



# 16진수 데이터 입력 후 10진수/2진수 출력

## ◆ 표준입력장치로 부터 입력 받을 때 16진수 형식으로 입력

- 입력 범위: 0x0000 ~ 0xFFFF
- 입력된 데이터의 값의 16진수, 10진수 및 2진수 표시
- 2진수 표시는 printInt\_Bits() 함수를 사용

## ◆ 입력된 2개의 데이터에 대한 산술연산 (+, -, \*, /, %)의 결과를 printIntOper\_Bits() 함수를 사용하여 표시

## ◆ 입력된 2개의 데이터에 대한 비트연산 (&, |, ^, ~, <<, >>)의 결과를 printIntOper\_Bits() 함수를 사용하여 표시



**printInt\_Bits()와 printIntOper\_Bits()**  
**함수들을 사용하는 main() 함수 구성**

# main() 함수

```
/* main() for integer arithmetic & bitwise operations (1) */
#include <stdio.h>

// preprocessor definitions
#define BIT_MASK 0x01
#define NUM_BITS_INT 32
enum BinOp_Type {INT_ADD, INT_SUB, INT_MUL, INT_DIV, INT_MOD, BIT_AND,
                 BIT_OR, BIT_XOR, BIT_NOT, SHIFT_LEFT, SHIFT_RIGHT };
const char *bop_sign[] = { "+", "-", "*", "/", "%", "&", "|", "^", "~", "<<", ">>" };

// function proto-types
void printInt_Bits(int d);
void printIntOper_Bits(BinOp_Type bop, int x, int y, int z);

void main(void)
{
    . . . . .
}

void printInt_Bits(int d)
{
    . . . . .
}

void printIntOper_Bits(BinOp_Type bop, int x, int y, int z)
{
    . . . . .
}
```



# main() 함수

```
/* main() for integer arithmetic & bitwise operations (2) */

void main ()
{
    int x, y, z;
    int k = 3;

    printf("Input two hexadecimal numbers in (0x0000 ~ 0xFFFF): ");
    scanf("%x %x", &x, &y);
    printf("Input hexadecimal numbers: \n");
    printf(" x = %#010X = %8d\n  = ", x, x); printInt_Bits(x); printf("\n");
    printf(" y = %#010X = %8d\n  = ", y, y); printInt_Bits(y); printf("\n");

    z = x + y;
    printf("\nInteger Addition : %#010X + %#010X => %#010X\n", x, y, z);
    printIntOper_Bits(INT_ADD, x, y, z);

    z = x - y;
    printf("\nInteger Subtraction : %#010X - %#010X => %#010X\n", x, y, z);
    printIntOper_Bits(INT_SUB, x, y, z);
}
```





# main() 함수

```
/* main() for integer arithmetic & bitwise operations (3) */
```

```
z = x * y;  
printf("\nInteger Multiplication : %d * %d => %d\n", x, y, z);  
printIntOper_Bits(INT_MUL, x, y, z);
```

```
z = x / y;  
printf("\nInteger Division : %d / %d => %d\n", x, y, z);  
printIntOper_Bits(INT_DIV, x, y, z);
```

```
z = x % y;  
printf("\nInteger Modulo : %d mod %d => %d\n", x, y, z);  
printIntOper_Bits(INT_MOD, x, y, z);
```

```
z = x & y;  
printf("\nBitwise AND : %#010X & %#010X => %#010X\n", x, y, z);  
printIntOper_Bits(BIT_AND, x, y, z);
```

```
z = x | y;  
printf("\nBitwise OR : %#010X | %#010X => %#010X\n", x, y, z);  
printIntOper_Bits(BIT_OR, x, y, z);
```



# main() 함수

```
/* main() for integer arithmetic & bitwise operations (4) */
```

```
z = x ^ y;  
printf("\nBitwise XOR : %#X ^ %#X => %#X\n", x, y, z);  
printIntOper_Bits(BIT_XOR, x, y, z);  
  
z = x << k;  
printf("\nBitwise Shift Left : %#X << %#X => %#X\n", x, k, z);  
printIntOper_Bits(SHIFT_LEFT, x, k, z);  
  
z = y >> k;  
printf("\nBitwise Shift Right : %#X >> %#X => %#X\n", y, k, z);  
printIntOper_Bits(SHIFT_RIGHT, y, k, z);  
  
z = ~y;  
printf("\nBitwise NOT : ~%#X => %#X\n", y, y, z);  
printIntOper_Bits(BIT_NOT, y, y, z);  
  
}
```



## 실습 2 문제 설명

# 프로그래밍언어 실습2

## ◆ 10 진수 숫자의 역순 출력

- 1) 최대 10자리수 이내의 10진수를 입력받고, 그 10진수 숫자를 역순으로 출력하는 프로그램의 pseudo code를 작성하라.
- 2) 위 pseudo code를 기반으로 C 프로그램을 작성하고, 정확한 실행결과를 확인하라.
- 3) Visual studio의 break point 설정 및 trace 기능을 사용하여, 위 C 프로그램의 중간 실행과 정에서의 지역 변수값을 확인하라.

<프로그램 실행 예>

```
Input positive decimal number (upto 10 digits): 123456789  
Digits in reverse order: 9 8 7 6 5 4 3 2 1  
계속하려면 아무 키나 누르십시오 . . .
```

```
Input positive decimal number (upto 10 digits): 345  
Digits in reverse order: 5 4 3  
계속하려면 아무 키나 누르십시오 . . .
```



## 프로그래밍언어 실습2

### ◆ 10진수 데이터의 8진수, 16진수, 2진수 표현과 Bit-wise 계산

- 1) 정수형 (int) 데이터의 값을 32비트 단위로 출력하는 함수 `printInt_Bits(int d)`를 작성하라.
- 2) `0x00000 ~ 0xFFFF` 범위의 값을 가지는 16진수 2개를 입력 받고, 이들 16진수 데이터 값의 8진수, 10진수 및 2진수 값을 출력하라.
- 3) 입력된 2개의 16진수 값의 덧셈과 뺄셈을 계산하고, 계산 결과값의 16진수 및 2진수 값을 각각 출력하라. `printIntOper_Bits()` 함수를 사용하여 출력.
- 4) 16진수 2개 (a, b)를 입력받아 bitwise AND ( `a & b` ), bitwise OR ( `a | b` ), bitwise XOR ( `a ^ b` ), `shift_left ( a << 3 )`, `shift_right ( b >> 3 )`, bitwise NOT ( `~b` )를 각각 계산하고, 계산 결과 값의 16진수 및 2진수 값을 각각 출력하라.



# 프로그래밍언어 실습2

## ◆ 프로그램 실행 예

```
Input first hexadecimal number in (0x00000000 ~ 0xFFFFFFFF): 0x1234ABCD
x = 0x1234ABCD = (in decimal) 305441741 = (in octal) 2215125715
  = (in bits) 00010010 00110100 10101011 11001101
Input second hexadecimal number in (0x00000000 ~ 0xFFFFFFFF): 0x87654321
y = 0x87654321 = (in decimal) -2023406815 = (in octal) 20731241441
  = (in bits) 10000111 01100101 01000011 00100001

Bitwise AND : 0x1234ABCD & 0x87654321 => 0x02240301
  00010010 00110100 10101011 11001101
& 10000111 01100101 01000011 00100001
-----
  00000010 00100100 00000011 00000001

Bitwise OR : 0x1234ABCD | 0x87654321 => 0x9775EBED
  00010010 00110100 10101011 11001101
| 10000111 01100101 01000011 00100001
-----
  10010111 01110101 11101011 11101101

Bitwise XOR : 0x1234ABCD ^ 0x87654321 => 0x9551E8EC
  00010010 00110100 10101011 11001101
^ 10000111 01100101 01000011 00100001
-----
  10010101 01010001 11101000 11101100

Bitwise Shift Left : 0x1234ABCD << 0x3 => 0x91A55E68
  00010010 00110100 10101011 11001101
<< 00000000 00000000 00000000 00000011
-----
  10010001 10100101 01011110 01101000

Bitwise Shift Right : 0x87654321 >> 0x3 => 0xF0ECA864
  10000111 01100101 01000011 00100001
>> 00000000 00000000 00000000 00000011
-----
  11110000 11101100 10101000 01100100

Bitwise NOT : ~0x87654321 => 0x87654321
  ~ 10000111 01100101 01000011 00100001
-----
  01111000 10011010 10111100 11011110
```



# Oral Test

- Q2.1 정수 (integer) 데이터를 10진수, 8진수, 16진수로 출력하기 위한 printf() 포맷 인 %d, %o, %x에 대하여 설명하라. (8진수는 0으로 시작하도록 하고, 16진수는 0x로 시작하도록 설정함)  
(**Key points**: 출력 공간, 소수점 이하 출력 자리 수, prefix 출력, 숫자 앞의 빈자리에 0 채우기 등 기능에 대한 설명)
- Q2.2 정수 (integer) 데이터를 2진수로 출력하기 위한 알고리즘 (printInt\_Bits(int data))의 pseudo code를 작성하고, 이에 대하여 설명하라. (**Key points**: int 자료형에 포함된 32개 비트를 차례로 출력할 수 있는 bitwise 연산 기능 사용)
- Q2.3 2진수 데이터의 bit-wise AND, bit-wise OR, bit-wise XOR 계산을 예를 들어 설명하라. 음수 (negative number)의 shift right에서 sign bit 부분이 어떻게 처리되는가에 대하여 설명하라. (**Key points**: printInt\_Bits() 함수를 사용)
- Q2.4 컴퓨터 시스템에서 2의 보수 (2's compliment)로 음의 정수 (음수, negative integer)값을 표현하는 방법에 대하여 설명하고, 정수형 변수 (integer variable)에 overflow가 발생하는 상황에 대하여 설명하라. (**Key points**: 2의 보수 (2's compliment)에 대하여 설명하고, <limits.h>에서 정의하고 있는 INT\_MAX의 비트 표현과 INT\_MAX + 1의 비트 패턴, INT\_MIN과 INT\_MIN - 1의 비트 패턴에 대하여 설명, printInt\_Bits() 함수를 사용하여 출력하여 확인.)

