

- ▶ 상수(심화)
- ▶ 데이터형(심화)
- ▶ scanf 함수
- ▶ 연산자



Artificial Intelligence Laboratory

## 상수(심화)

## 리터럴(Literal) 상수

- ▶ 값 자체.
- ▶ 예) 1, '1', 1.0

## 매크로(Macro) 상수

- ▶ `#define`을 사용하여 정의
- ▶ 예) `#define ONE 1`

## `const` 변수

- ▶ `const` 키워드를 사용
- ▶ 변수를 상수와 같은 수정 불가능한 상태로 만들어 줌

# 리터럴(Literal 상수)

I 값 자체.

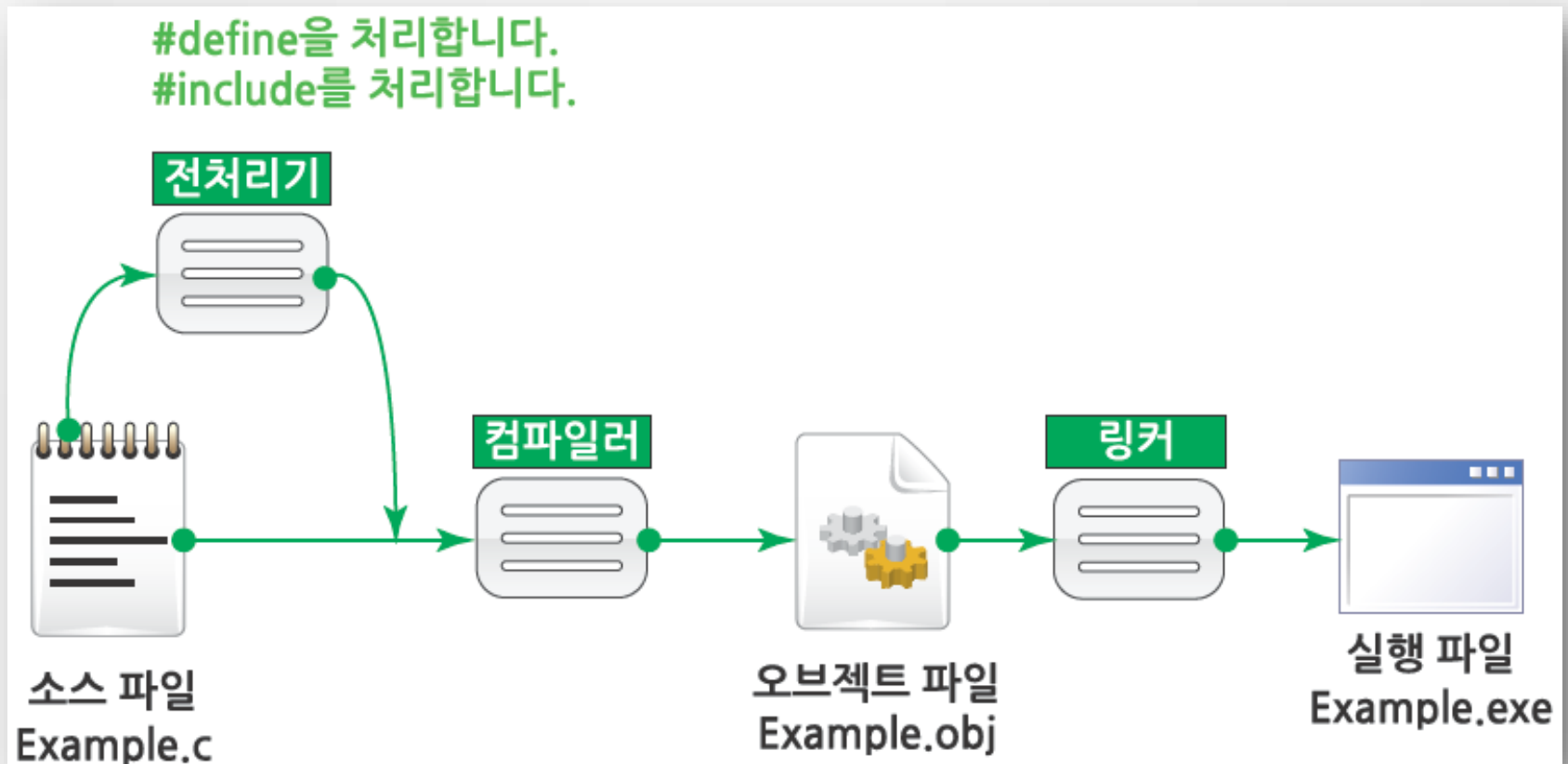
## ■ #define문으로 정의되는 상수

형식	#define	매크로명	값
예제	#define	BUF_SIZE	80
	#define	MAX	100
	#define	MIN	-MAX
	#define	PI	3.1415
	#define	MSG	"잘못 입력했습니다."
		매크로명	값

# 전처리기(preprocessor)

컴파일 과정은 사실 '전처리'와 '컴파일' 두 가지 작업을 거친다

- #include와 #define은 전처리 과정 때 처리가 된다.



# #define문의 처리

- #define문은 #define으로 정의된 매크로 상수를 특정 값으로 치환(replace)

## 전처리기 수행 전

```
#define MAX 100

int main(void)
{
    int a = MAX;
    int b = MAX - 1;
    printf("MAX = %d\n", MAX);
    ...
}
```

## 전처리기 수행 후

```
int main(void)
{
    int a = 100;
    int b = 100 - 1;
    printf("MAX = %d\n", 100);
    ...
}
```

# 매크로 상수 정의 시 주의사항!!!

- #define문은 세미콜론(;)이 필요 없다. (C와 문법이 다르기 때문)

```
#define MAX 100;
int num2 = MAX - 1;
```

매크로 정의의 끝에 ;을 사용  
int num2=100;-1;으로 처리된다.  
문장 중간에 ;이 들어가므로 컴파일 에러

- 매크로 상수의 값은 변경할 수 없다.

```
MAX = 200;
```

100 = 200;으로 처리된다.  
매크로 상수는 변경할 수 없으므로 컴파일 에러



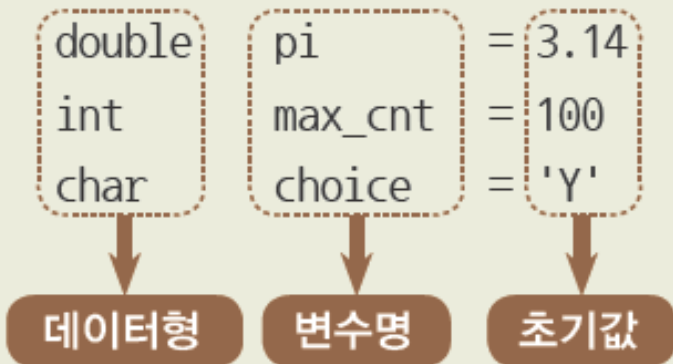
# const 변수

- 변수 선언 시 데이터형 앞에 **const**를 지정하면 상수가 된다
  - const** 변수는 반드시 선언과 함께 초기화가 이루어져야 한다

**형식**     `const 데이터형 변수명 = 초기값;`

**예제**

```
const double pi = 3.14 ;
const int max_cnt = 100 ;
const char choice = 'Y' ;
```



데이터형     변수명     초기값

# const 변수

## const 는 변수값 수정을 불가능하게 만든다

- 시도하면 컴파일 에러가 난다
- 굳이 사용하지 않아도 아무 문제는 없다. 하지만 원활한 협업을 위해 꼭 필요하다.

const double pi = 3.14 ; ○ ————— const 변수 선언

pi = 3.1415 ; ○ ————— const 변수는 변경할 수 없으므로 컴파일 에러

# const 변수의 선언 및 사용

```
01: /* Ex03_04.c */
02: #include <stdio.h>
03:
04: int main(void)
05: {
06:     const double pi = 3.14; ← const 변수 선언
07:     const int max; ← 초기화 되지 않았으므로 쓰레기 값을 가짐
08:
09:     printf("pi = %f\n", pi);
10:     printf("max = %d\n", max);
11:
12:     pi = 3.1415; ← const 변수는 변경할 수 없다.
13:     max = 100;
14:
15:     return 0;
16: }
```

## 실행 결과

```
pi = 3.140000
max = -858993460
```

Artificial Intelligence Laboratory

# 데이터형(심화)

## ■ 기본 데이터형(primitive data type)

- ◆ 문자형 : char
- ◆ 정수형 : short, int, long, long long
- ◆ 실수형 : float, double

## ■ 파생 데이터형(derived data type)

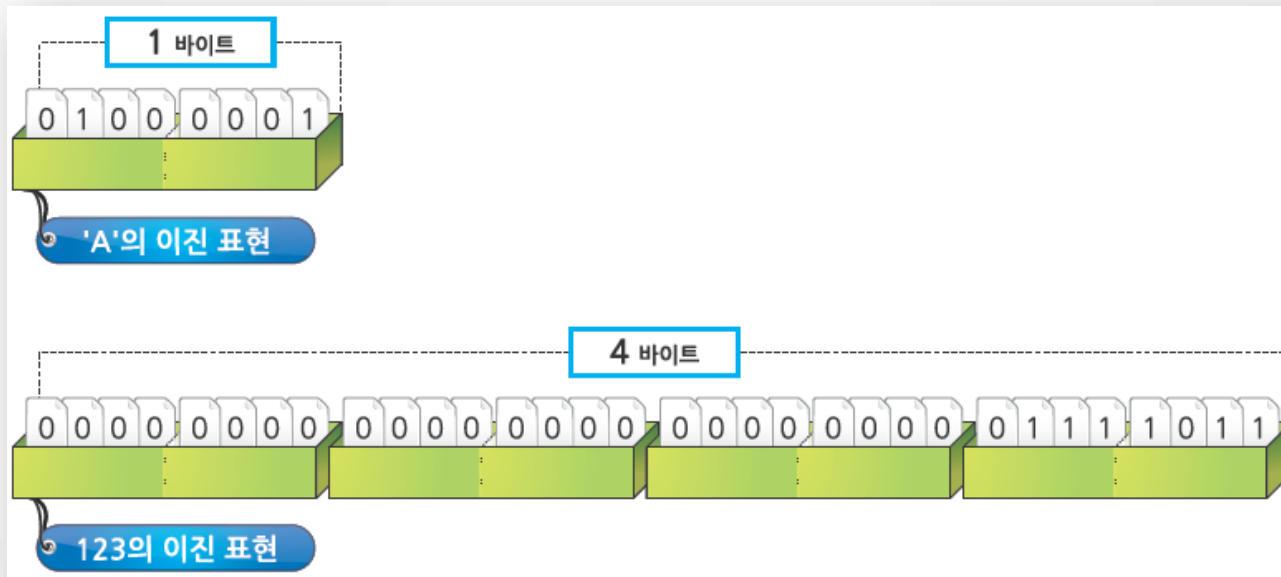
- ◆ 배열(array), 포인터(pointer)

## ■ 사용자 정의형(user-defined data type)

- ◆ 구조체(struct), 공용체(union), 열거체(enum)

# 데이터형(Data Type)

- 동일한 이진수라도 데이터형에 따라 해석이 다르다

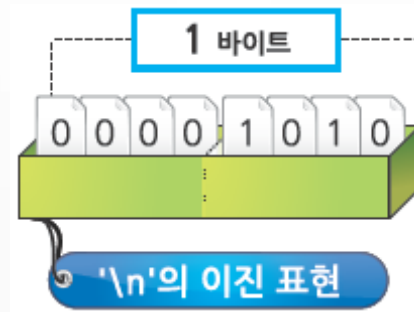


# 정수의 표현

- 부호 있는 정수형은 최상위 비트(MSB)를 **부호 비트**로 사용함
- 부호 있는 정수형은 **음수를 표현하는 데 2의 보수를 사용함**



- C 언어는 문자형으로 **char형**을 제공함
- Char형은 **1바이트 크기**의 데이터형으로 **문자 코드를 저장함**
- 대표적인 문자 코드로는 ASCII 코드나 한글 완성형 코드 등이 있음
- 특수 문자
  - ▶ ASCII 코드 중 특별한 용도로 사용되는 특수 문자는 '\와 함께 나타냄





# 특수문자와 코드

특수 문자	10진수 코드	16진수 코드	이름
'\0'	0	00	널 문자(null)
'\a'	7	07	경고음(bell)
'\b'	8	08	백스페이스(backspace)
'\t'	9	09	수평탭(horizontal tab)
'\n'	10	0A	줄바꿈(newline)
'\v'	11	0B	수직탭(vertical tab)
'\f'	12	0C	폼 피드(form feed)
'\r'	13	0D	캐리지 리턴(carriage return)
'\"'	34	22	큰따옴표
'\''	39	27	작은따옴표
'\\'	92	5C	역슬래시(back slash)

# 데이터형의 유효범위

분류	데이터형	바이트 크기	유효 범위
문자형	char	1	$-128(-2^7) \sim 127(2^7-1)$
	unsigned char	1	$0 \sim 255(2^8-1)$
정수형	short	2	$-32768(-2^{15}) \sim 32767(2^{15}-1)$
	unsigned short	2	$0 \sim 65535(2^{16}-1)$
	int	4	$-2147483648(-2^{31}) \sim 2147483647(2^{31}-1)$
	unsigned int	4	$0 \sim 4294967295(2^{32}-1)$
	long	4	$-2147483648(-2^{31}) \sim 2147483647(2^{31}-1)$
	unsigned long	4	$0 \sim 4294967295(2^{32}-1)$
실수형	float	4	$\pm 1.17549 \times 10^{-38} \sim \pm 3.40282 \times 10^{38}$
	double	8	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{308}$
	long double	8	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{308}$

Artificial Intelligence Laboratory

# sizeof 연산자

# sizeof 연산자

- 주어진 값이나 데이터형의 바이트 크기를 구할 때 사용

형식

```
sizeof 값  
sizeof (값)  
sizeof (데이터형)
```

예제

```
sizeof 123  
sizeof num1  
sizeof ( num1 )  
sizeof ( num1 + 100 )
```

값

```
sizeof ( int )  
sizeof ( double )
```

데이터형

# sizeof 연산자의 사용 예

```
16: printf("char  형의 바이트 크기 : %d\\n", sizeof(char));
17: printf("short 형의 바이트 크기 : %d\\n", sizeof(short));
18: printf("int   형의 바이트 크기 : %d\\n", sizeof(int));
19: printf("long  형의 바이트 크기 : %d\\n", sizeof(long));
20: printf("float  형의 바이트 크기 : %d\\n", sizeof(float));
21: printf("double 형의 바이트 크기 : %d\\n", sizeof(double));
22:
23: return 0;
24: }
```

## 실행 결과

```
ch      의 바이트 크기  :  1
num     의 바이트 크기  :  4
fnum    의 바이트 크기  :  8
3.14f   의 바이트 크기  :  4
char    형의 바이트 크기 :  1
short   형의 바이트 크기 :  2
int     형의 바이트 크기 :  4
long    형의 바이트 크기 :  4
float   형의 바이트 크기 :  4
double  형의 바이트 크기 :  8
```

Artificial Intelligence Laboratory

# scanf 함수

# 입력을 위한 scanf 함수

```
int main(void)
{
    int num;
    scanf("%d", &num);
    . . . .
}
```

변수 num 에 저장하라.

scanf( "%d", &num );

10진수 정수형태로 입력 받아서

- printf 함수에서의 %d는 10진수 정수의 출력을 의미한다.
- 반면 scanf 함수에서의 %d는 10진수 정수의 입력을 의미한다.
- 변수의 이름 num 앞에 & 를 붙인 이유는 이후에 천천히 알게 된다.

```
int main(void)
{
    int result;
    int num1, num2;

    printf("정수 one: ");
    scanf("%d", &num1);    // 첫 번째 정수 입력
    printf("정수 two: ");
    scanf("%d", &num2);    // 두 번째 정수 입력

    result=num1+num2;
    printf("%d + %d = %d \n", num1, num2, result);
    return 0;
}
```

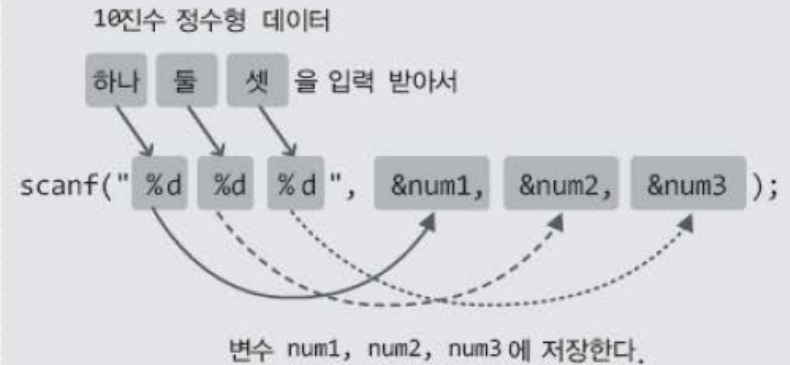
## 실행결과

정수 one: 3  
정수 two: 4  
3 + 4 = 7

# 입력의 형태는 다양하게 지정할 수 있음

```
int main(void)
{
    int num1, num2, num3;
    scanf("%d %d %d", &num1, &num2, &num3);
    . . . .
}
```

한 번의 `scanf` 함수호출을 통해서 둘 이상의 데이터를 원하는 방식으로 입력 받을 수 있다.



```
int main(void)
{
    int result;
    int num1, num2, num3;
    printf("세 개의 정수 입력: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    result=num1+num2+num3;
    printf("%d + %d + %d = %d \n", num1, num2, num3, result);
    return 0;
}
```

실행결과

세 개의 정수 입력: 4 5 6  
4 + 5 + 6 = 15



# 실습 과제 1

- | 저장 폴더: 학번\_이름\_5주차
- | 파일 이름: area.c
- | 사각형의 넓이를 구하는 프로그램을 작성하시오. (단, 사용자로부터 가로와 세로의 길이를 입력 받아야 함)

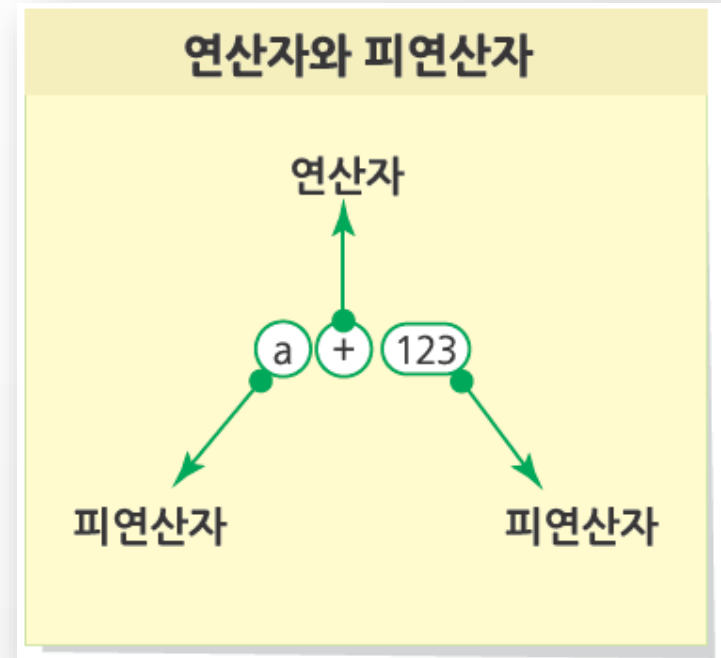
# 연산자와 피연산자

## 연산자

- 연산에 사용되는 기호
- $+$ ,  $-$ ,  $++$ ,  $--$ ,  $==$ ,  $>$ ,  $<$ ,  $\&\&$ ,  $\|$ ,  $<<$ ,  $>>$  등

## 피연산자

- 연산의 대상이 되는 값



# 대입 연산자와 산술 연산자

연산자	연산자의 기능	결합방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) num = 20;	←
+	두 피연산자의 값을 더한다. 예) num = 4 + 3;	→
-	왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다. 예) num = 4 - 3;	→
*	두 피연산자의 값을 곱한다. 예) num = 4 * 3;	→
/	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다. 예) num = 7 / 3;	→
%	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다. 예) num = 7 % 3;	→

```
int main(void)
{
    int num1=9, num2=2;
    printf("%d+%d=%d \n", num1, num2, num1+num2);
    printf("%d-%d=%d \n", num1, num2, num1-num2);
    printf("%d×%d=%d \n", num1, num2, num1*num2);
    printf("%d÷%d의 몫=%d \n", num1, num2, num1/num2);
    printf("%d÷%d의 나머지=%d \n", num1, num2, num1%num2);
    return 0;
}
```

함수호출 문장에 연산식이 있는 경우  
연산이 이뤄지고 그 결과를 기반으로 함수의 호출이  
진행된다.

9 + 2 = 11

9 - 2 = 7

9 × 2 = 18

9 ÷ 2의 몫 = 4

9 ÷ 2의 나머지 = 1

실행결과

# 복합 대입 연산자



```
int main(void)
{
    int num1=2, num2=4, num3=6;
    num1 += 3;    // num1 = num1 + 3;
    num2 *= 4;    // num2 = num2 * 4;
    num3 %= 5;    // num3 = num3 % 5;
    printf("Result: %d, %d, %d \n", num1, num2, num3);
    return 0;
}
```

실행결과

Result: 5, 16, 1

# 증가, 감소 연산자

연산자	연산자의 기능	결합방향
++num	값을 1 증가 후, 속한 문장의 나머지를 진행(선 증가, 후 연산) 예) val = ++num;	←
num++	속한 문장을 먼저 진행한 후, 값을 1 증가(선 연산, 후 증가) 예) val = num++;	←
--num	값을 1 감소 후, 속한 문장의 나머지를 진행(선 감소, 후 연산) 예) val = --num;	←
num--	속한 문장을 먼저 진행한 후, 값을 1 감소(선 연산, 후 감소) 예) val = num--;	←

```
int main(void)
{
    int num1=12;
    int num2=12;
    printf("num1: %d \n", num1);
    printf("num1++: %d \n", num1++); // 후위 증가
    printf("num1: %d \n\n", num1);
    printf("num2: %d \n", num2);
    printf("++num2: %d \n", ++num2); // 전위 증가
    printf("num2: %d \n", num2);
    return 0;
}
```

```
num1: 12
num1++: 12
num1: 13

num2: 12
++num2: 13
num2: 13
```

실행결과

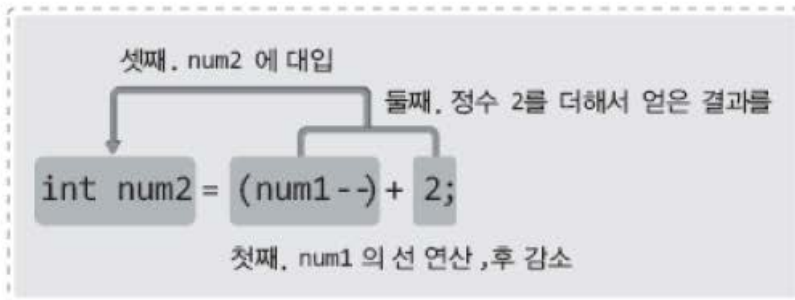
# 증가, 감소 연산자

```
int main(void)
{
    int num1=10;
    int num2=(num1--)+2;    // 후위 감소
    printf("num1: %d \n", num1);
    printf("num2: %d \n", num2);
    return 0;
}
```

num1: 9

num2: 12

실행결과

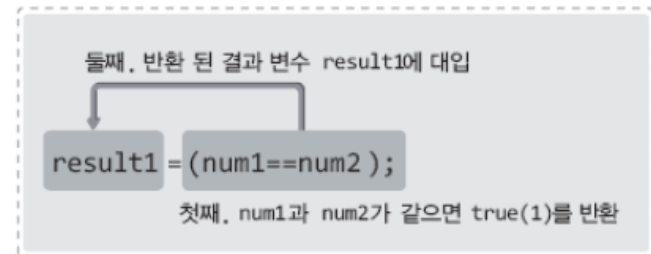


연산의 과정

# 관계 연산자

연산자	연산자의 기능	결합방향
<	예) $n1 < n2$ n1이 n2보다 작은가?	→
>	예) $n1 > n2$ n1이 n2보다 큰가?	→
==	예) $n1 == n2$ n1과 n2가 같은가?	→
!=	예) $n1 != n2$ n1과 n2가 다른가?	→
<=	예) $n1 <= n2$ n1이 n2보다 같거나 작은가?	→
>=	예) $n1 >= n2$ n1이 n2보다 같거나 큰가?	→

연산의 조건을 만족하면 참을 의미하는 1을 반환하고  
만족하지 않으면 거짓을 의미하는 0을 반환하는 연산  
자들이다.



C언어는 0이 아닌 모든 값을 참으로 간주한다. 다만  
1이 참을 의미하는 대표적인 값일 뿐이다.

실행결과

```

result1: 0
result2: 1
result3: 0
  
```

```

int main(void)
{
    int num1=10;
    int num2=12;
    int result1, result2, result3;
    result1=(num1==num2);
    result2=(num1<=num2);
    result3=(num1>num2);
    printf("result1: %d \n", result1);
    printf("result2: %d \n", result2);
    printf("result3: %d \n", result3);
    return 0;
}
  
```

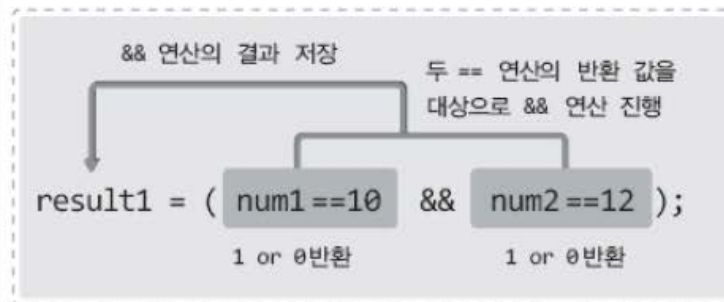
# 논리 연산자

연산자	연산자의 기능	결합방향
&&	예) A && B A와 B 모두 '참'이면 연산결과로 '참'을 반환(논리 AND)	→
	예) A    B A와 B 둘 중 하나라도 '참'이면 연산결과로 '참'을 반환(논리 OR)	→
!	예) !A A가 '참'이면 '거짓', A가 '거짓'이면 '참'을 반환(논리 NOT)	←

```
int main(void)
{
    int num1=10;
    int num2=12;
    int result1, result2, result3;
    result1 = (num1==10 && num2==12);
    result2 = (num1<12 || num2>12);
    result3 = (!num1);
    printf("result1: %d \n", result1);
    printf("result2: %d \n", result2);
    printf("result3: %d \n", result3);
    return 0;
}
```

result1: 1  
result2: 1  
result3: 0

실행결과



왼쪽 예제에서 num1은 0이 아니므로 참과 거짓의 관계로 본다면 거짓에 해당한다. 따라서 ! 연산의 결과로 참을 의미하는 1이 반환되는 것이다.



# 연산자의 우선순위와 결합방향

## ✓ 연산자의 우선순위

- 연산의 순서에 대한 순위
- 덧셈과 뺄셈보다는 곱셈과 나눗셈의 우선순위가 높다.

## ✓ 연산자의 결합방향

- 우선순위가 동일한 두 연산자 사이에서의 연산을 진행하는 방향
- 덧셈, 뺄셈, 곱셈, 나눗셈 모두 결합방향이 왼쪽에서 오른쪽으로 진행된다.

$$3 + 4 \times 5 \div 2 - 10$$

연산자의 우선순위에 근거하여 곱셈과 나눗셈이 먼저 진행된다.

결합방향에 근거하여 곱셈이 나눗셈보다 먼저 진행된다.

## 실습 과제 2

- | 저장 폴더: 학번\_이름\_5주차
- | 파일 이름: clock.c
- | 시, 분, 초를 입력 받아서 총 몇 초인지 계산하는 프로그램을 작성하세요.(단, 정상적인 범위의 숫자만 들어온다고 가정함)