

프로그래밍언어

2. 변수, 상수, 배열과 연산



교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ 변수 (variable) 와 상수 (constant)
- ◆ 기호 상수 (symbolic constant)
- ◆ 자료형 (data type)
- ◆ 기본 자료형의 배열 (array)
- ◆ 수식과 연산자 (operator)란?
- ◆ 산술연산자, 대입연산자
- ◆ 관계연산자, 논리연산자, 조건연산자, 콤마연산자
- ◆ 비트단위 연산자
- ◆ 연산자의 우선순위 (precedence)와 결합 규칙 (association rule)
- ◆ Homework 2



**변수 (variable)와 상수 (constant),
기호 상수 (symbolic constant)**

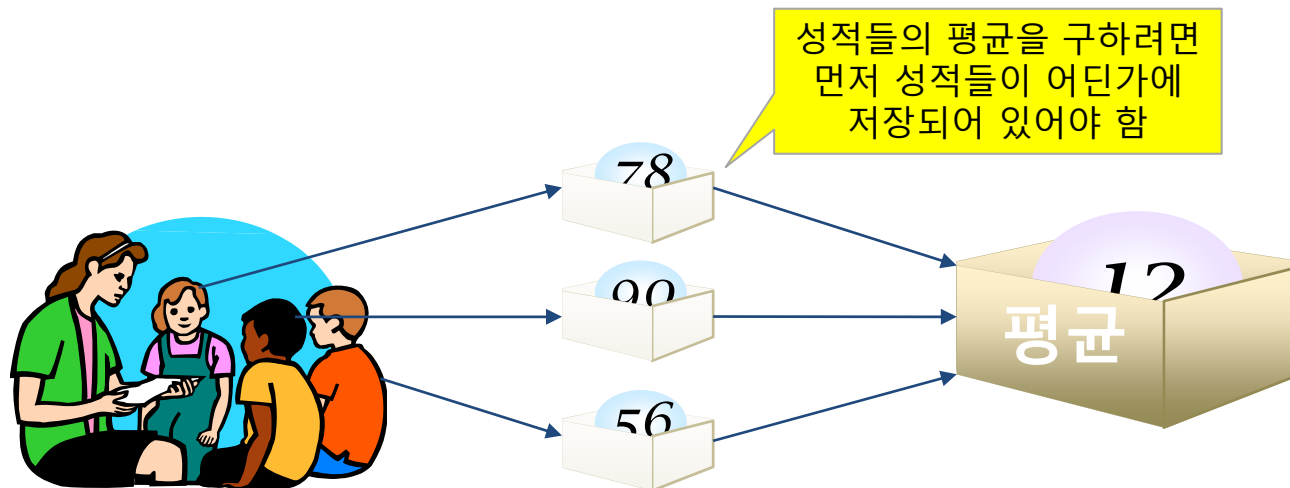
변수 (Variable)

Q) 변수(variable)이란 무엇인가?

- 프로그램에서 일시적으로 데이터를 저장하는 공간
- 변수에 저장되어 있는 값은 변경될 수 있음

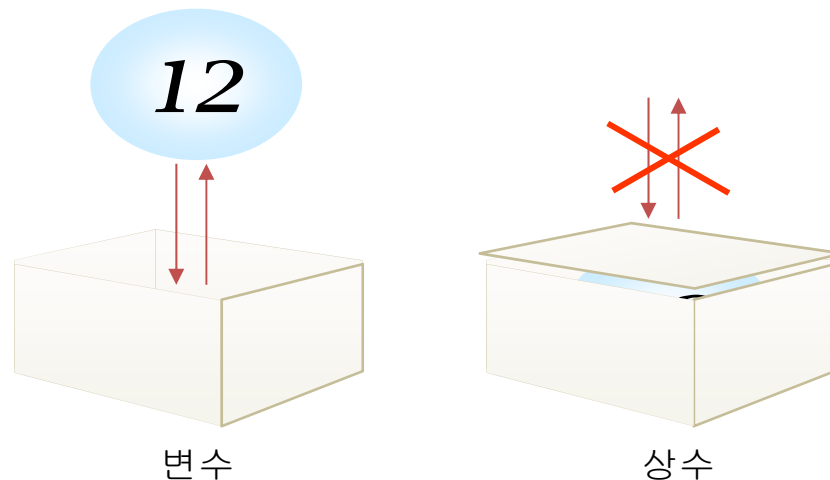
Q) 변수는 왜 필요한가?

- 데이터가 입력되면 그 값을 어딘가에 저장해 두어야만 다음에 사용할 수 있다.



변수와 상수

- ◆ **변수(variable)**: 프로그램 실행 중에 저장된 값의 변경이 가능한 수; 데이터를 보존하고 필요 시에 참조하며 관리하기 위하여 변수를 사용
- ◆ **상수(constant)**: 사용자가 한번 설정한 후, 프로그램 실행 중에는 저장된 값의 변경이 불가능한 수
(예) 3.14, 100, 'A', "Hello World!"



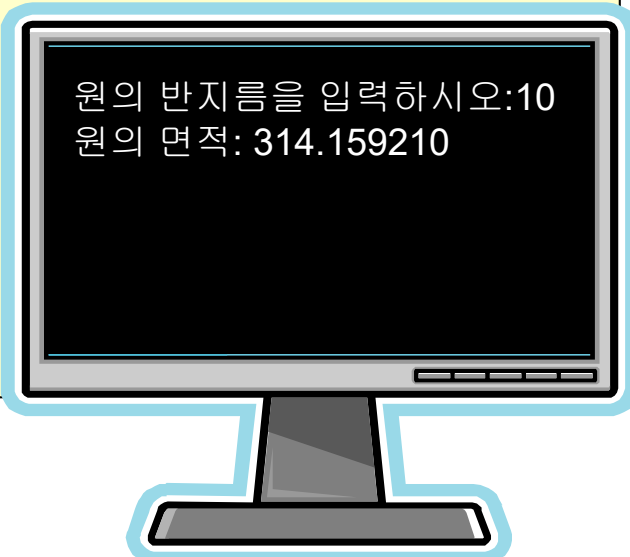
예제: 변수와 상수

```
/* 원의 면적을 계산하는 프로그램 */ 상수
#include <stdio.h>
#define PI 3.141592 변수
int main(void)
{
    double radius; // 원의 반지름
    double area; // 원의 면적

    printf("원의 반지름을 입력하시오:");
    scanf("%lf", &radius);

    area = PI * radius * radius;
    printf("원의 면적: %lf \n", area);

    return 0;
}
```



원의 반지름을 입력하시오:10
원의 면적: 314.159210



자료형

◆ 자료형(data type): 데이터의 타입(종류)

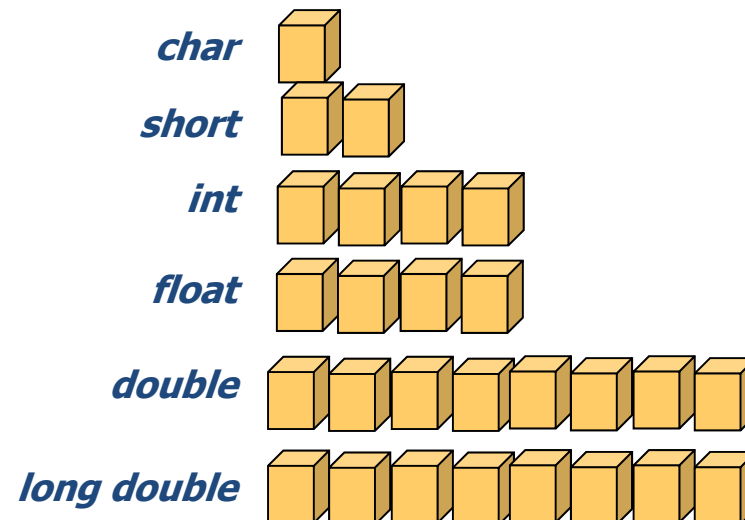
(예) short, int, long: 정수형 데이터(100)

(예) double, float: 실수형 데이터(3.141592)

(예) char: 문자형 데이터('A', 'a', '한')

◆ 자료형의 크기 확인

- sizeof(char)
- sizeof(int)
- sizeof(double)



변수의 이름짓기

◆ 식별자(identifier): 식별할 수 있게 해주는 이름

- 변수 이름
- 함수 이름

◆ 식별자 작성 원칙

- 알파벳 문자와 숫자, 밑줄 문자 _로 구성
- 첫 번째 문자는 반드시 알파벳 또는 밑줄 문자 _
- 대문자와 소문자를 구별
- C 언어의 키워드와 똑같은 이름은 허용되지 않는다.

(Q) 다음은 유효한 식별자인가?

sum

_count

king3

n_pictures

2nd_try // 숫자로 시작

Dollar# // 특수 문자 포함

double // C 언어 keyword



키워드 (Keyword)

- ◆ 키워드(keyword): C언어에서 고유한 의미를 가지고 있는 특별한 단어
- ◆ 예약어(reserved word) 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



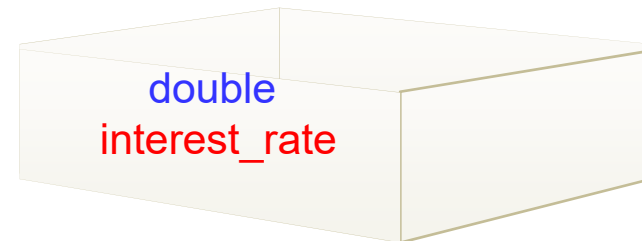
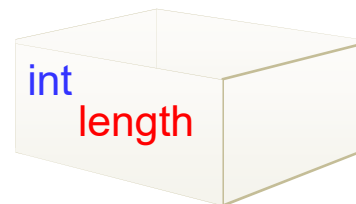
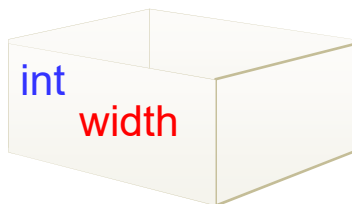
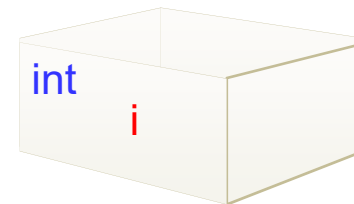
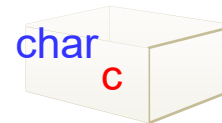
변수 선언

- ◆ 변수 선언: 컴파일러에게 어떤 변수를 사용하겠다고 미리 알리는 것

자료형 변수이름;

- ◆ 변수 선언의 예

- `char c;`
- `int i;`
- `double interest_rate;`
- `int width, length;`

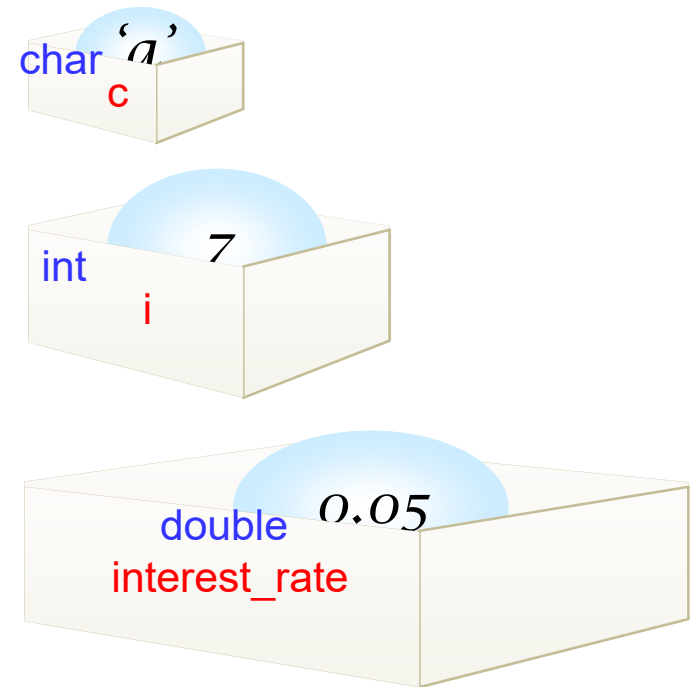


변수의 초기화

◆ 변수 초기화

- 변수를 선언하면서 초기값을 설정

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



예제: 변수의 선언

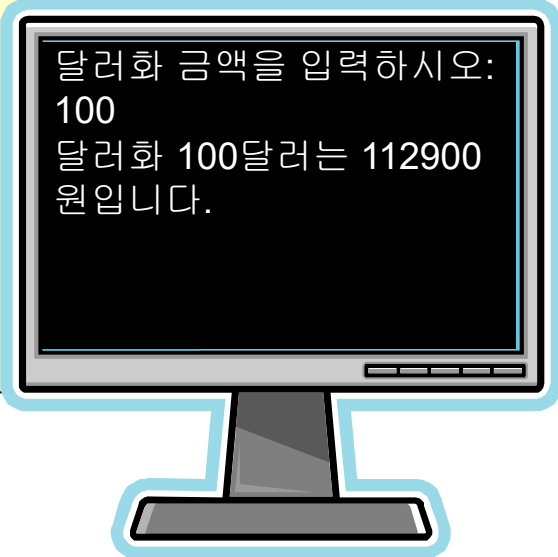
```
#include <stdio.h>
#define EXCHANGE_RATE 1129 // Korean Won per 1 US Dollar
int main(void)
{
    int usd; // 달러화
    int krw; // 원화

    printf("달러화 금액을 입력하시오: ");
    scanf("%d", &usd);

    krw = EXCHANGE_RATE * usd;
    printf("달러화 %d달러는 %d원입니다.", usd, krw);

    return 0;
}
```

변수 선언



달러화 금액을 입력하시오:
100
달러화 100달러는 112900
원입니다.

112900

krw

=

EXCHANGE_
RATE (1129)

*

100

usd



기호 상수 (Symbolic Constant)

◆ 기호 상수(symbolic constant): 기호를 이용하여 상수를 표현한 것

◆ (예)

- $\text{area} = 3.141592 * \text{radius} * \text{radius};$

- $\text{area} = \text{PI} * \text{radius} * \text{radius};$

- $\text{income} = \text{salary} - 0.15 * \text{salary};$

- $\text{income} = \text{salary} - \text{TAX_RATE} * \text{salary};$

◆ 기호 상수의 장점

- 가독성 (readability)이 높아진다.

- 값을 쉽게 변경할 수 있다.



정수 상수 (integer constant)

◆ 숫자를 적으면 기본적으로 int형이 된다.

- num = 123; // 123은 int형

◆ 상수의 자료형을 명시하려면 다음과 같이 한다.

- NUM = 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: 기호 상수

```
/** Calculation of the Area of Circle */
#include <stdio.h>
#define PI 3.141592653589793
int main(void)
{
    double radius = 0;
    double area = 0.0, circumference = 0.0;
    printf("Input radius of a circle (in double) : ");
    scanf("%lf", &radius);
    area = radius * radius * PI;
    circumference = 2.0 * radius * PI;
    printf("circle of radius (%lf)\n", radius);
    printf("radius = %lf\n", area);
    printf("circumference = %lf\n", circumference);
}
```



자료형 (data type)

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	-2147483648 ~ 2147483647
		long	long형 정수	(32)	-2147483648 ~ 2147483647
	부호 없음	unsigned short	부호 없는 short형 정수	2 (16)	0 ~ 65535
		unsigned int	부호 없는 정수	4	0 ~ 4294967295
		unsigned long	부호 없는 long형 정수	(32)	0 ~ 4294967295
실수형 (부동 소수점형)		float	단일정밀도 부동소수점	4 (32)	1.2E-38 ~ 3.4E38
		double	두배 정밀도 부동소수점	8	2.2E-308 ~ 1.8E308
		long double	두배 정밀도 부동소수점	(64)	2.2E-308 ~ 1.8E308



문자형(char)

- ◆ 문자는 컴퓨터보다는 인간에게 중요
- ◆ 문자도 컴퓨터 내부에서는 숫자를 이용하여 표현
- ◆ 다양한 국가에서 공통으로 사용할 수 있는 국제 규격이 필요하다.
- ◆ 아스키 코드(ASCII: American Standard Code for Information Interchange)
 - 8 비트를 사용하여 영어 알파벳 표현
 - (예) '!'는 33, 'A'는 65, 'B'는 66, 'a'는 97, 'b'는 98

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



ASCII (Character Set)

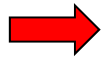
◆ ASCII (American Standard Code for Information Interchange)

	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



문자 변수 (character variable)

```
/* 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);  
}
```



(Q) 1과 '1'의 차이점은?

(A) 1은 정수 상수 (integer constant) 이고,
'1'은 문자 상수 (character constant) 이다.



제어 문자 (Control Character)

◆ 인쇄 목적이 아니라 제어 (control) 목적으로 사용되는 문자들

- (예) 줄 바꿈 (new line) 문자, 탭 (tab) 문자, 벨 (bell) 소리 문자, 백스페이스 (back space) 문자

◆ 제어 문자를 나타내는 방법

- 아스키 코드를 직접 사용

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

- 이스케이프 시퀀스 사용

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



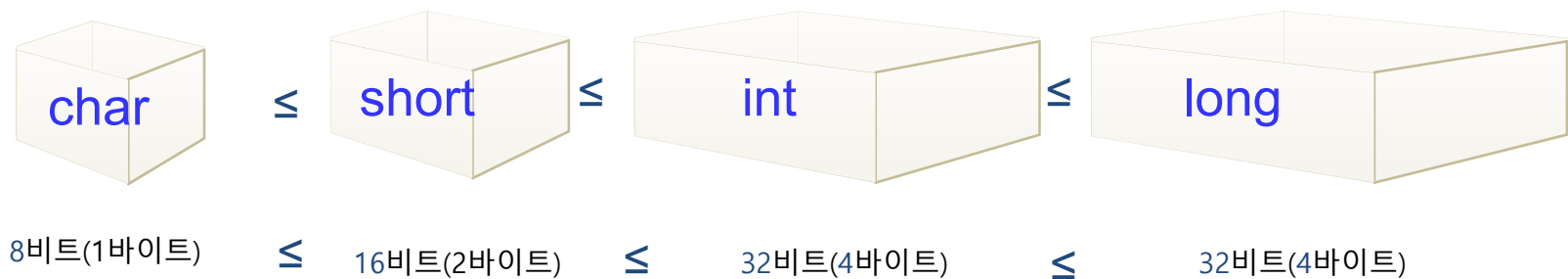
Escape Sequence

제어 문자 이름	제어 문자 표기	값	의미
NULL 문자	\0	0	문자열의 끝을 표시
alarm(bell)	\a	7	"삐"하는 경고 벨소리 발생
백스페이스(backspace)	\b	8	커서를 현재의 위치에서 한 글자 뒤로 옮긴다.
수평 탭(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 (double quotation mark)	\"	34	원래의 큰따옴표 자체
작은따옴표 (single quotation mark)	\'	39	원래의 작은따옴표 자체
역슬래시(back slash)	\\	92	원래의 역슬래시 자체



정수형 (integer)

◆ 정수 자료형: **char, short, int, long**



- 가장 기본이 되는 것은 int
 - CPU에 따라서 크기가 달라진다: 8비트, 16비트, 32비트, 64비트
 - 8비트 char도 정수형 데이터로 사용 가능

(Q) 왜 여러 개의 정수형이 필요한가?

(A) 프로그램의 용도에 따라 사용되는 값의 범위를 가장 작은 메모리 공간으로 표현할 수 있는 자료형을 프로그래머가 선택하여 사용할 수 있게 하여 메모리 공간을 효율적으로 사용하기 위하여
예) 차량의 색상 종류가 100가지 미만일 때: int 대신 char 사용 가능

정수형의 범위

◆ char 형

-128, -127, , -1, 0, 1, 2, 127

char형의 범위:
총 256 (-128 ~
+ 127)

◆ short 형

$-2^{15}, \dots, -2, -1, 0, 1, 2, \dots, 2^{15} - 1$
-32768 ~ +32767

◆ int형

$-2^{31}, \dots, -2, -1, 0, 1, 2, \dots, 2^{31} - 1$
-2147483648 ~ +2147483647

int형의 범위:
약 -21억 ~ +21억

◆ long 형

보통 int형과 같음

64비트 운영체제의 경우 long형을 64비트로 표현



부동소수점 실수형 (float, double)

◆ float

- 단일 정밀도 부동소수점
- 4 bytes (32 bits)
- $1.2 \times 10^{-38} \sim 3.4 \times 10^{38}$

◆ double

- 2배 정밀도 부동소수점
- 8 bytes (64 bits)
- $2.2 \times 10^{-308} \sim 1.8 \times 10^{308}$

◆ long double

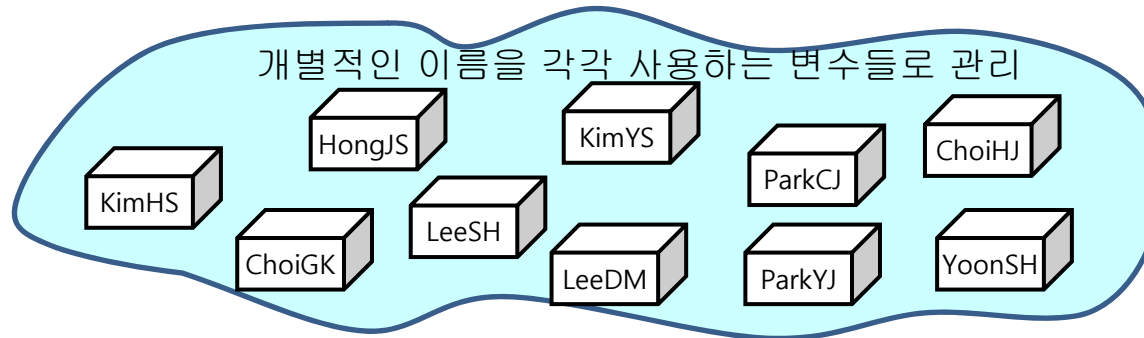
- 2배 정밀도 부동소수점
- 8 bytes (64 bits)
- $2.2 \times 10^{-308} \sim 1.8 \times 10^{308}$



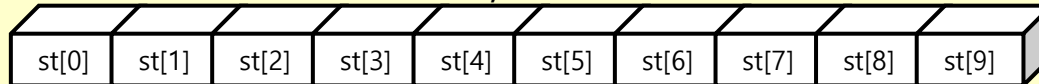
기본 자료형의 배열 (Array)

배열이란?

◆ 여러 개의 데이터를 관리하는 방법



하나의 이름으로 배열을 관리, 개수가 증가해도 동일한 배열이름사용



- ◆ **배열(array):** 동일한 자료형의 데이터가 여러 개 저장되어 있는 데이터 저장 장소
- ◆ 배열 안에 들어있는 각각의 데이터들은 정수로 되어 있는 인덱스(첨자)에 의하여 접근
- ◆ 배열을 이용하면 동일한 자료형의 여러 개의 데이터 들을 하나의 이름으로 관리할 수 있다.

배열의 필요성

- 학생이 100명이 있고 이들의 평균 성적을 계산한다고 가정하자.

개별 변수를 사용하는
방법은 학생수가
 많아지면
 번거로워집니다.



방법 1 : 개별 변수 사용

```
int s0;  
int s1;  
...  
int s99;
```

방법 2 : 배열 사용

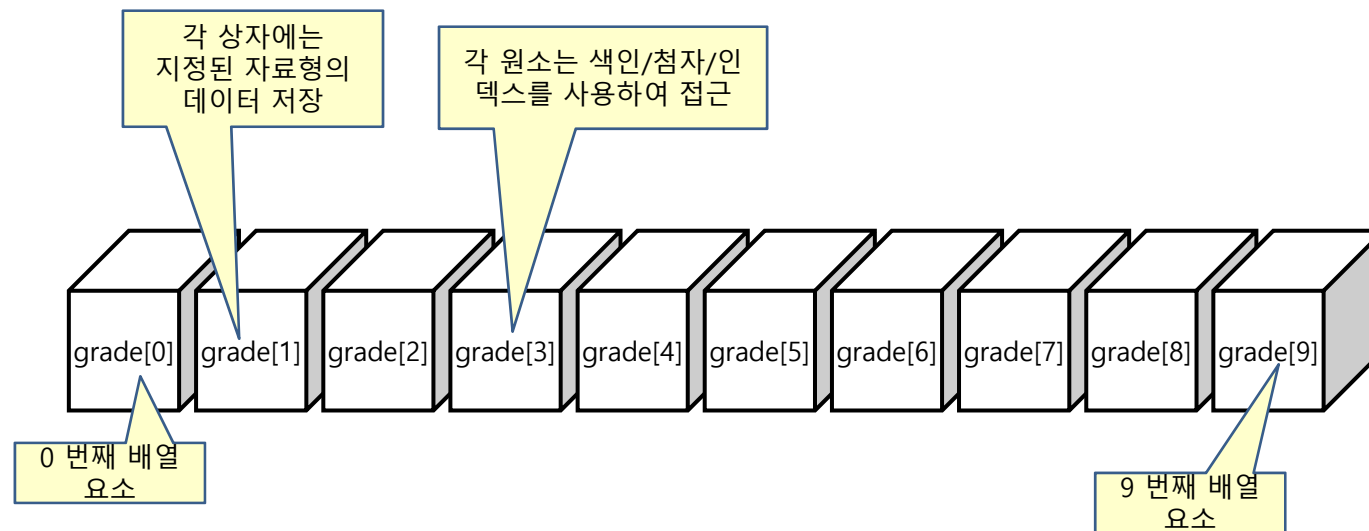
```
int s[100];  
s[0] = 95;  
...  
s[99] = 85;
```



배열 원소와 인덱스

◆ 색인, 인덱스(index)

- 배열 원소의 색인 번호
- 총 N개의 배열 원소가 포함되어 있을 경우 인덱스는 0 ~ N-1



배열의 선언

```
int grade[100];
```

자료형 배열이름 배열크기
 (원소의 개수)

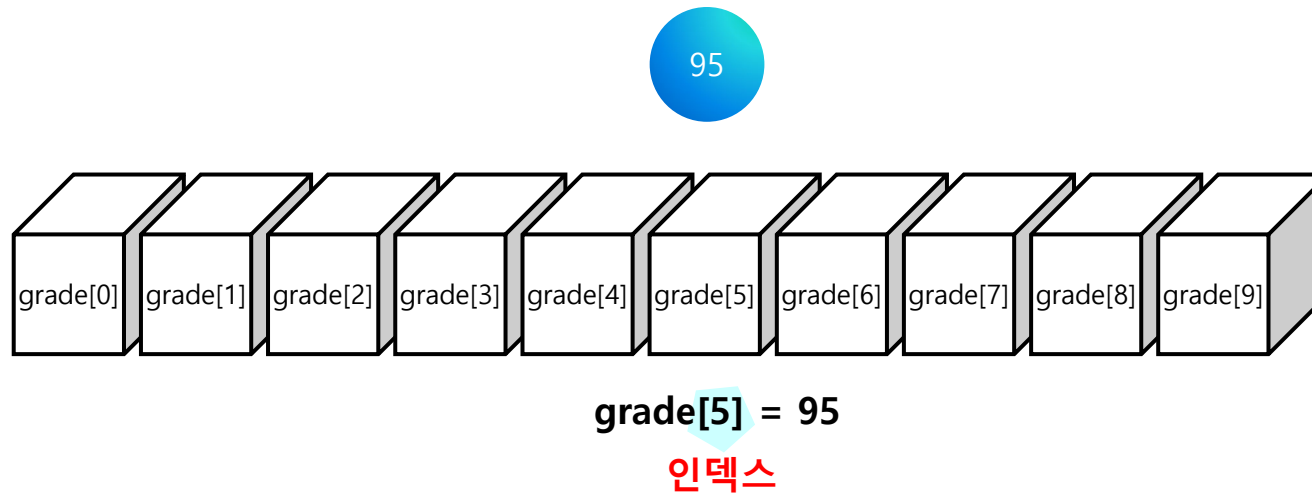
- ◆ 자료형: 배열 원소들이 **int**형라는 것을 의미
- ◆ 배열 이름: 배열을 사용할 때 사용하는 이름이 **grade**
- ◆ 배열 크기: 배열 원소의 개수가 **100**개
- ◆ 인덱스(배열 번호)는 항상 0부터 시작한다.

```
int grade[60];           // 60개의 int형 값을 가지는 배열 grade
double cost[12];         // 12개의 double형 값을 가지는 배열 cost
char name[50];           // 50개의 char형 값을 가지는 배열 name
char src[10], dst[10];   // 2개의 문자형 배열을 동시에 선언
int index, days[7];      // 일반 변수와 배열을 동시에 선언
```



배열 원소 접근

- 배열 이름과 인덱스를 사용하여 배열 원소에 접근



```
grade[5] = 95;
grade[1] = grade[0];
grade[i] = 100;      // i는 정수 변수
grade[i+2] = 100;    // 수식이 인덱스가 된다.
grade[index[3]] = 100; // index[]는 정수 배열
```

배열 선언 예제

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
```

```
int main(void)
{
    int i;
    int grade[SIZE];
```

```
    for(i = 0; i < SIZE; i++)
        grade[i] = rand() % 100;
```

```
    for(i = 0; i < SIZE; i++)
        printf("grade[%d]=%d\n", i, grade[i]);
```

```
    return 0;
```

```
}
```

```
grade[0]=41
grade[1]=67
grade[2]=34
grade[3]=0
grade[4]=69
```



배열 선언 예제

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int main(void)
{
    int i;
    int grade[SIZE];
    printf("5명의 점수를 입력하시오\n");

    for(i = 0; i < SIZE; i++)
        scanf("%d", &grade[i]);

    for(i = 0; i < SIZE; i++)
        printf("grade[%d]=%d\n", i, grade[i]);
    return 0;
}
```

5명의 점수를 입력하시오
23 35 67 45 21
grade[0]=23
grade[1]=35
grade[2]=67
grade[3]=45
grade[4]=21



배열 선언 예제



```
#include <stdio.h>
#define STUDENTS 5
int main(void)
{
    int grade[STUDENTS];
    int i;
    double sum=0.0, average;
    for(i = 0; i < STUDENTS; i++)
    {
        printf("학생들의 성적을 입력하시오: ");
        scanf("%d", &grade[i]);
    }
    for(i = 0; i < STUDENTS; i++)
        sum += grade[i];
    average = sum / STUDENTS;
    printf("성적 평균 = %lf\n", average);

    return 0;
}
```



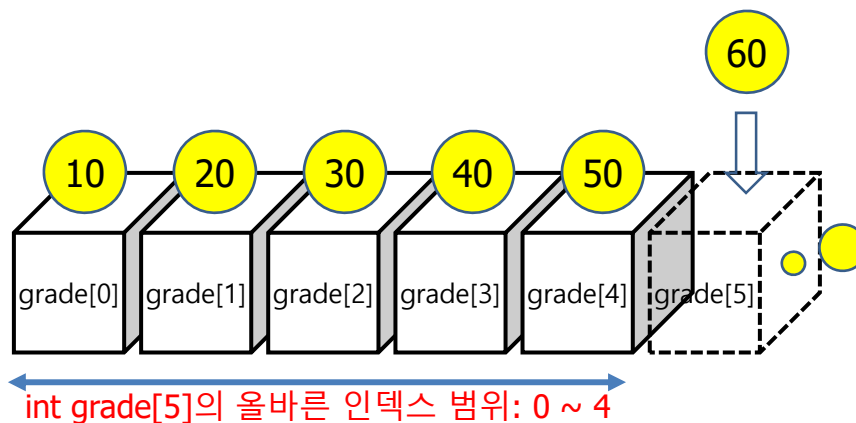
학생들의 성적을 입력하시오: 10
학생들의 성적을 입력하시오: 20
학생들의 성적을 입력하시오: 30
학생들의 성적을 입력하시오: 40
학생들의 성적을 입력하시오: 50
성적 평균 = 30.0



잘못된 인덱스 문제

- ◆ 인덱스가 배열의 크기를 벗어나게 되면 프로그램에 치명적인 오류를 발생시킨다.
- ◆ C에서는 프로그래머가 인덱스가 범위를 벗어나지 않았는지를 확인하고 책임을 져야 한다.

```
int grade[5]; // index: 0 ~ 4  
...  
grade[5] = 60; // 치명적인 오류!
```



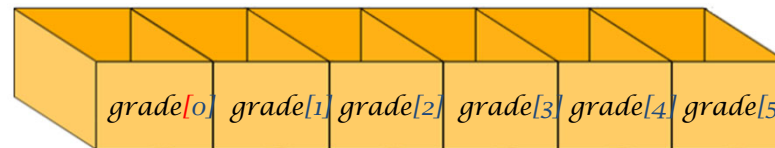
배열의 크기에 따라 한정되는 인덱스의 올바른 범위를 벗어난 곳에 데이터를 저장하려고 하면 치명적인 오류가 발생합니다.



배열의 초기화

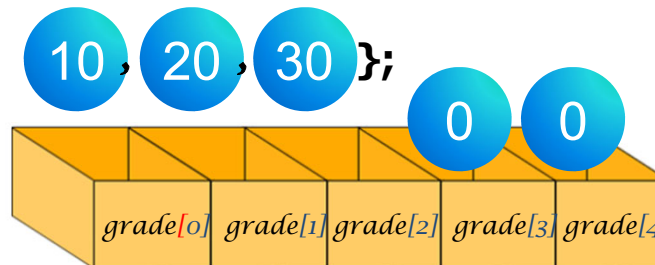
◆ `int grade[5] = { 10,20,30,40,50 };`

`int grade[5] = {` 10, 20, 30, 40, 50, 60 `};`



◆ `int grade[5] = { 10,20,30 };`

`int grade[5] = {` 10, 20, 30 `};`



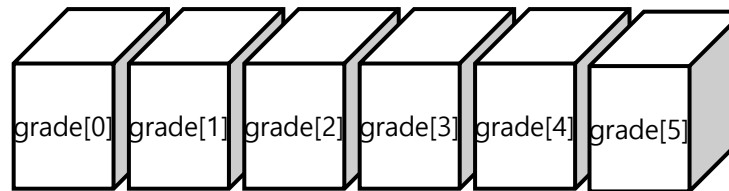
초기값을 일부
만 주면 나머지
원소들은 0으로
초기화됩니다.



배열의 초기화

- ◆ 배열의 크기가 주어지지 않으면 자동적으로 초기값의 개수만큼이 배열의 크기로 잡힌다.

`int grade[] = { 10, 20, 30, 40, 50, 60 };`



배열 초기화 예제

```
#include <stdio.h>
int main(void)
{
    int grade[5] = { 31, 63, 62, 87, 14 };
    int i;

    for(i = 0; i < 5; i++)
        printf("grade[%d] = %d\n", i, grade[i]);

    return 0;
}
```

```
grade[0] = 31
grade[1] = 63
grade[2] = 62
grade[3] = 87
grade[4] = 14
```



배열의 복사

```
int grade[SIZE];  
int score[SIZE];
```

```
score = grade; // 컴파일 오류!
```

잘못된 방법

```
#include <stdio.h>  
#define SIZE 5
```

```
int main(void)  
{
```

```
    int i;
```

```
    int a[SIZE] = {1, 2, 3, 4, 5};
```

```
    int b[SIZE];
```

```
    for(i = 0; i < SIZE; i++)  
        b[i] = a[i];
```

```
    return 0;
```

```
}
```

올바른 방법

원소를 일일이
복사한다



배열의 비교

```
#include <stdio.h>
#define SIZE 5

int main(void)
{
    int i;
    int a[SIZE] = { 1, 2, 3, 4, 5 };
    int b[SIZE] = { 1, 2, 3, 4, 5 };

    if( a == b )                // ① 올바른지 않은 배열 비교
        printf("잘못된 결과입니다.\n");
    else
        printf("잘못된 결과입니다.\n");

    for(i = 0; i < SIZE ; i++) // ② 올바른 배열 비교
    {
        if ( a[i] != b[i] )
        {
            printf("a[]와 b[]는 같지 않습니다.\n");
            return 0;
        }
    }
    printf("a[]와 b[]는 같습니다.\n");
    return 0;
}
```

원소를 개별적으로
각각 비교한다



배열 원소 역순 출력



```
#include <stdio.h>
#define SIZE 5
```

```
int main(void)
{
```

```
    int data[SIZE];
    int i;
```

```
    for(i = 0; i < SIZE; i++)
```

```
    {
        printf("정수를 입력하시오:");
        scanf("%d", &data[i]);
```

```
    }
```

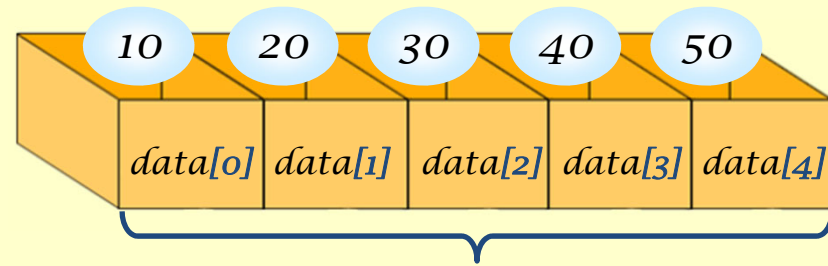
```
    for(i = SIZE - 1; i >= 0; i--)
```

```
    {
        printf("%d\n", data[i]);
```

```
    }
```

```
    return 0;
```

```
}
```



SIZE = 5

// 정수를 입력받는 루프

// 역순

```
정수를 입력하시오:10
정수를 입력하시오:20
정수를 입력하시오:30
정수를 입력하시오:40
정수를 입력하시오:50
50
40
30
20
10
```



주사위면 빈도 계산

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

#define SIZE 6

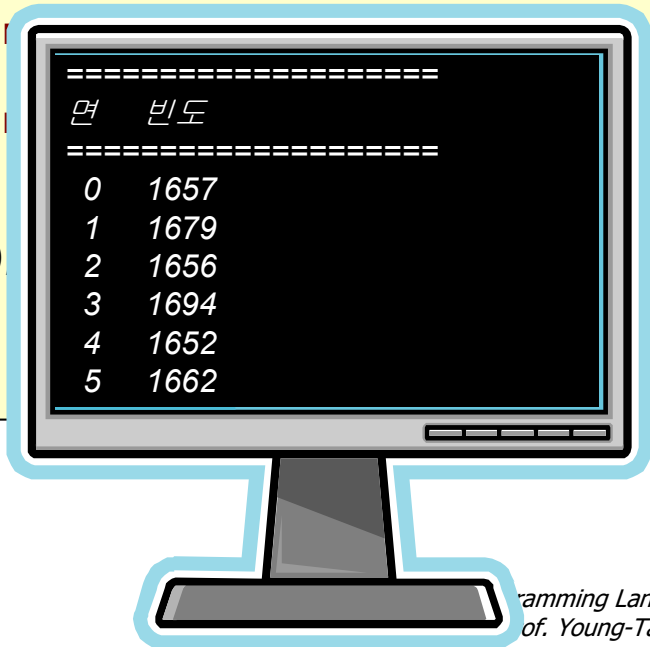
int main(void)
{
    int freq[SIZE] = { 0 };           // 주사위의 면의 빈도를 0으로 한다.
    int i;

    for(i = 0; i < 10000; i++)        // 주사위를 10000번 던진다.
        ++freq[ rand() % 6 ];        // 해당면의 빈도를 하나 증가한다.

    printf("=====\\n");
    printf("면   빈도\\n");
    printf("=====\\n");

    for(i = 0; i < SIZE; i++)
        printf("%3d   %3d \\n", i, freq[i]);

    return 0;
}
```



면	빈도
0	1657
1	1679
2	1656
3	1694
4	1652
5	1662

**연산 (operation), 연산자 (operator),
피연산자 (operand)**

수식 (Expression)

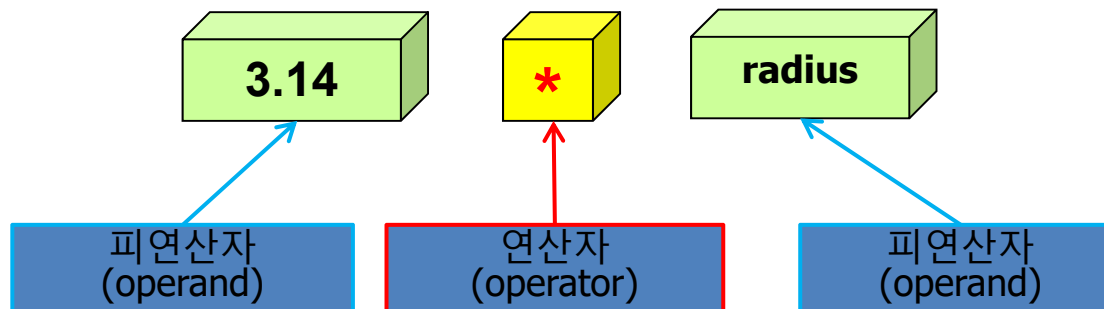
◆ 수식(expression)

$x + y$

$x * x + 5 * x + 6$

$(\text{principal} * \text{interest_rate} * \text{period}) / 12.0$

- 수식은 상수, 변수, 연산자의 조합
- 연산자 (operator)와 피연산자 (operand)로 나누어진다.



기능에 따른 연산자의 분류

연산자의 분류	연산자	의미
대입	=	오른쪽을 왼쪽에 대입
산술	+ - * / %	사칙연산과 나머지 연산
부호	+ -	
증감	++ --	증가, 감소 연산
관계	> < == != >= <=	오른쪽과 왼쪽을 비교
논리	&& !	논리적인 AND, OR
조건	?	조건에 따라 선택
coma	,	피연산자 들을 순차적으로 실행
비트 단위 연산자	& ^ ~ << >>	비트별 AND, OR, XOR, 반전, 이동(shift)
sizeof 연산자	sizeof	자료형이나 변수의 크기를 바이트 단위로 반환
형변환	(type)	변수나 상수의 자료형을 변환
포인터 연산자	* & []	주소계산, 포인터가 가리키는 곳의 내용 추출
구조체 연산자	. ->	구조체의 멤버 참조



피연산자수에 따른 연산자 분류

◆ 단항 연산자 (unary operator): 피연산자의 수가 1개

```
++x;  
--y;
```

◆ 이항 연산자 (binary operator) : 피연산자의 수가 2개

```
x + y  
x - y
```

◆ 삼항 연산자 (ternary operator): 연산자의 수가 3개

```
x ? y : z
```



산술 연산자

◆ 덧셈, 뺄셈, 곱셈, 나눗셈 등의 사칙 연산을 수행하는 연산자

연산자	기호	의미	예
덧셈	+	x와 y를 더한다	$x + y$
뺄셈	-	x에서 y를 뺀다.	$x - y$
곱셈	*	x와 y를 곱한다.	$x * y$
나눗셈	/	x를 y로 나눈다.	x / y
나머지	%	x를 y로 나눌 때의 나머지값	$x \% y$

$$y = mx + b$$

$$y = ax^2 + bx + c$$

$$m = \frac{x+y+z}{3}$$

$$y = m * x + b$$

$$y = a * x * x + b * x + c$$

$$m = (x + y + z) / 3$$

(참고) 거듭 제곱 연산자는?

C에는 거듭 제곱을 나타내는 연산자는 없다.
 $x * x$ 와 같이 단순히 변수를 두 번 곱한다.



증감 (increment/decrement) 연산자

증감 연산자	의미, 예제
++X	x값을 먼저 증가시킨 후, 다른 연산 (예: 대입연산)에 사용한다. x = 10; y = ++x; // pre-increment // x : 11, y : 11
X++	x값을 먼저 현재 상태로 사용한 후, 증가시킨다. x = 10; y = x++; // post-increment // y : 10, x : 11
--X	x값을 먼저 감소시킨 후, 다른 연산(예: 대입연산)에 사용한다. x = 10; y = --x; // pre-decrement // x : 9, y : 9,
X--	x값을 먼저 현재 상태로 사용한 후, 감소시킨다. x = 10; y = x--; // post-decrement // y : 10, x : 9

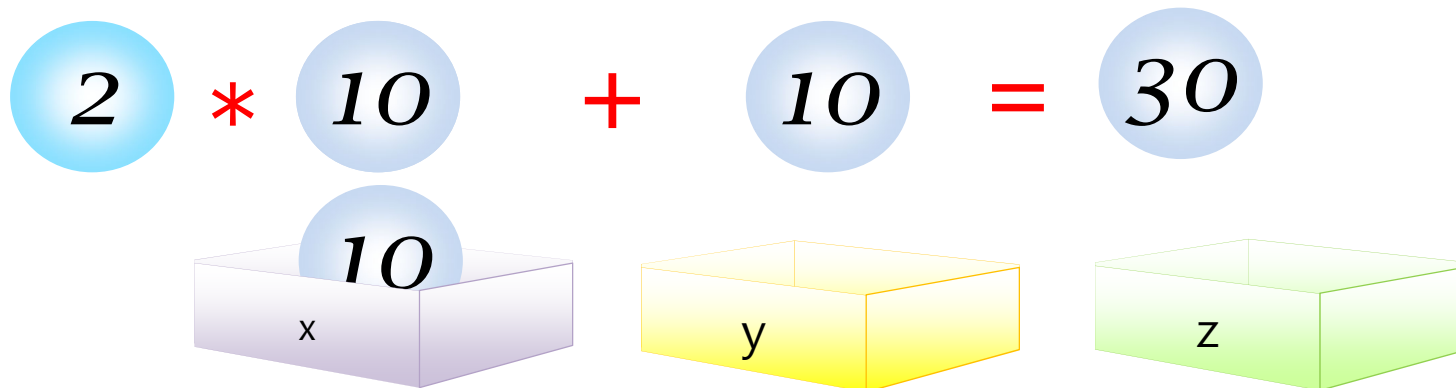
대입 연산자 (assignment operator)

◆ 왼쪽에 있는 변수에 오른쪽의 수식의 값을 계산하여 대입

변수(variable) = 수식(expression);



```
x = 10;    // 상수 10을 변수 x에 대입한다.  
y = x;     // 변수 x의 값을 변수 y에 대입한다.  
z = 2 * x + y; // 수식 2 * x + y를 계산하여 변수 z에 대입한다.
```



복합 대입 연산자

◆ 복합 대입 연산자란 +=처럼 대입연산자 =와 산술연산자 (+, -, *, /, % 등)를 합쳐 놓은 연산자

◆ 소스를 간결하게 만들 수 있음

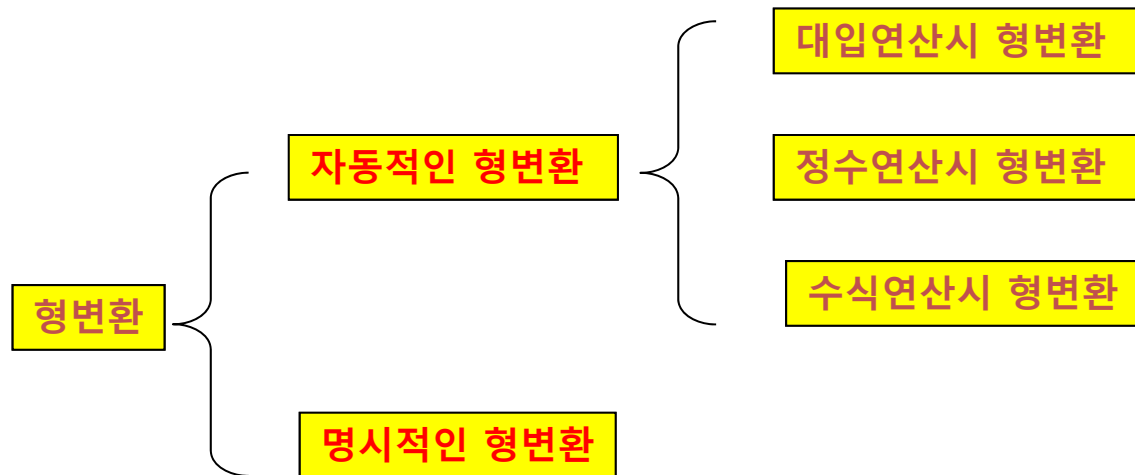
복합 대입 연산자	의미
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x / y$
$x \% = y$	$x = x \% y$
$x \& = y$	$x = x \& y$
$x = y$	$x = x y$
$x \wedge = y$	$x = x \wedge y$
$x \gg = y$	$x = x \gg y$
$x \ll = y$	$x = x \ll y$

```
x += 1      // x = x + 1
x * = y + 1  // x = x * (y + 1)
x \% = x + y  // x = x \% (x + y)
```



자료형 변환 (type conversion)

◆ 연산시에 데이터의 유형이 변환되는 것



변수의 자료형은 컴파일 단계와 프로그램 실행 단계에서 자동으로 변환되기도 하고, 사용자가 명시적으로 변경시키기도 합니다 !



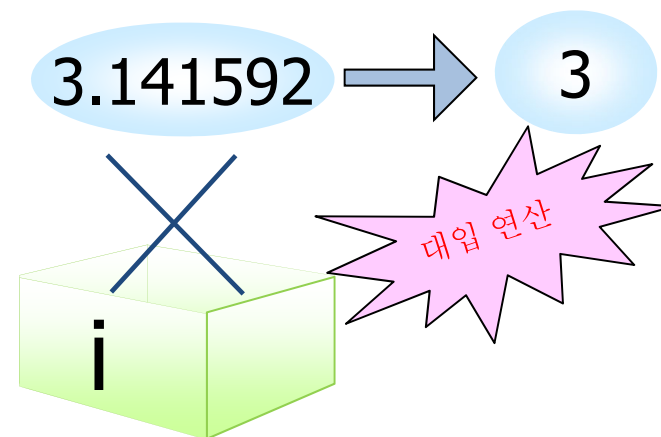
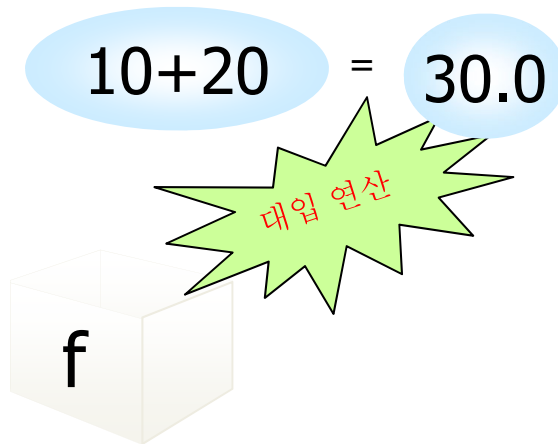
대입 연산시의 자동적인 형변환

◆ 올림 변환

```
double f;  
f = 10 + 20;           // f에는 30.0이 저장된다.
```

◆ 내림 변환

```
int i;  
i = 3.141592;          // i에는 3이 저장된다.
```



올림 변환과 내림 변환

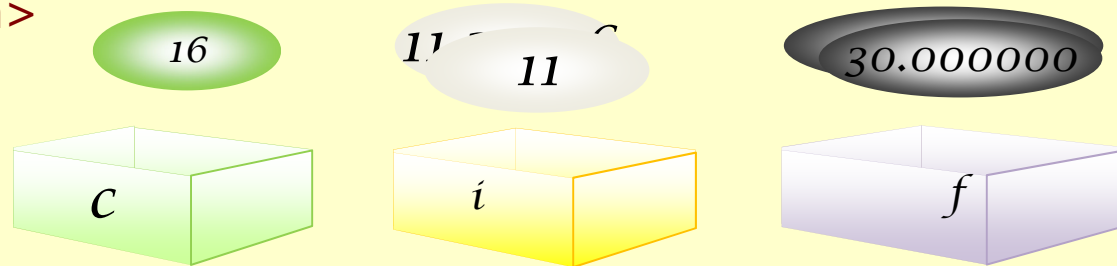
```
#include <stdio.h>
int main(void)
```

```
{
```

```
    char c;
    int i;
    double f;
```

```
    c = 10000;           // 내림 변환
    i = 1.23456 + 10;    // 내림 변환
    f = 10 + 20;         // 올림 변환
    printf("c = %d, i = %d, f = %lf \n", c, i, f);
    return 0;
```

```
}
```



```
c:\W...Wconvert1.c(10) : warning C4305: '=' : 'int'에서 'char'(으)로 잘립니다.
```

```
c:\W...Wconvert1.c(11) : warning C4244: '=' : 'double'에서 'int'(으)로 변환하면서 데이터가 손실될 수 있습니다.
```

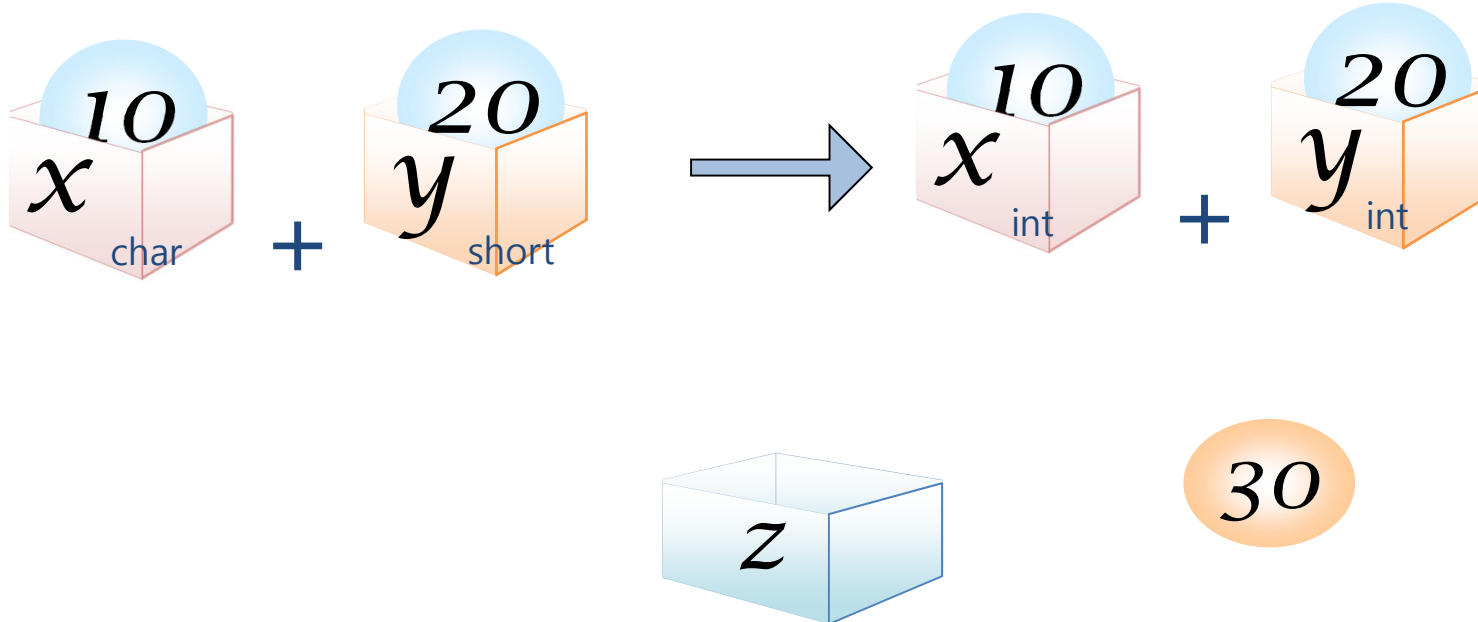
```
c=16, i=11, f=30.000000
```



정수 연산시의 자동적인 형변환

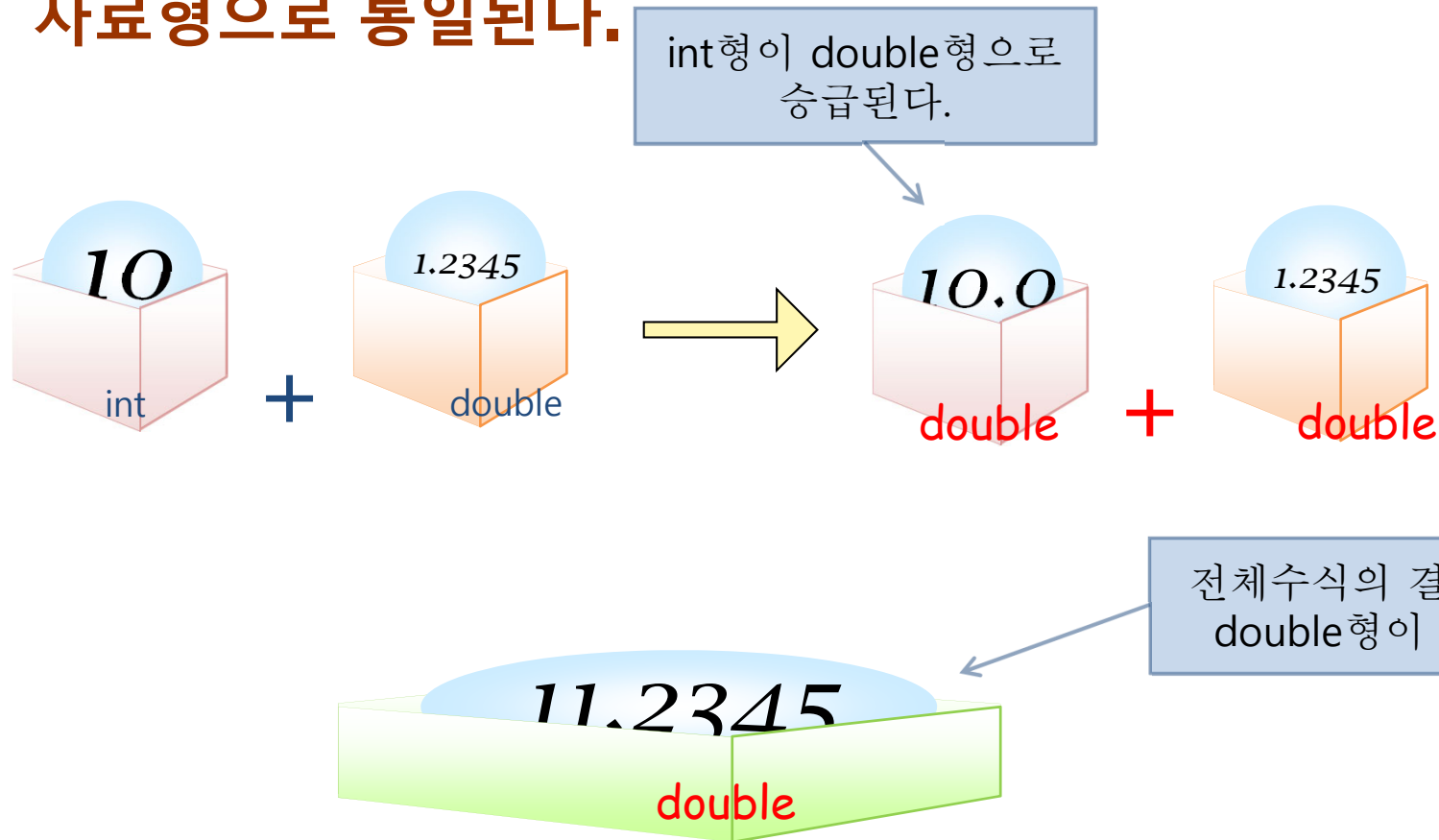
- ◆ 정수 연산시 char형이나 short형의 경우, 자동적으로 int형으로 변환하여 계산한다.

```
char x = 10;  
short y = 20;  
z = x + y;
```



수식에서의 자동적인 형변환

- ◆ 서로 다른 자료형이 혼합하여 사용되는 경우, 더 큰 자료형으로 통일된다.

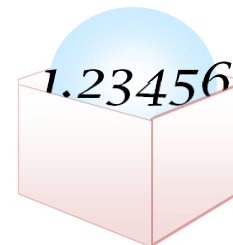
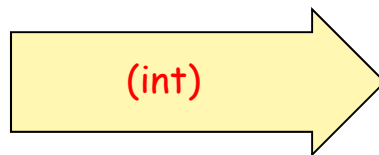
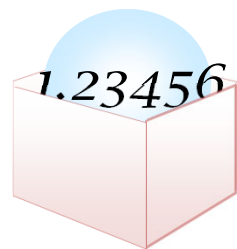


명시적인 형변환

- ◆ 형변환(type cast): 사용자가 데이터의 타입을 변경하는 것

(자료형) 상수 또는 변수

- ◆ (int) 1.23456
- ◆ (double) x // double형으로 변환
- ◆ (long) (x + y) // long형으로 변환



**관계연산자 (relational operator),
논리연산자 (logical operator),
조건연산자 (conditional operator),
비트 단위 연산자 (bit-wise operator)
연산자의 우선순위**

관계 연산자

- ◆ 두개의 피연산자를 비교하는 연산자
- ◆ 결과값은 참(1) 아니면 거짓(0)

수학 기호	연산자 기호	의미	사용예
$x \equiv y$	$x == y$	x와 y가 같은가?	$x == y$
$x \neq y$	$x != y$	x와 y가 다른가?	$x != y$
$x > y$	$x > y$	x가 y보다 큰가?	$x > y$
$x < y$	$x < y$	x가 y보다 작은가?	$x < y$
$x \geq y$	$x >= y$	x가 y보다 크거나 같은가?	$x >= y$
$x \leq y$	$x <= y$	x가 y보다 작거나 같은가?	$x <= y$



주의할 점!

◆ $(x = y)$

- x 의 값을 y 에 대입한다. 이 수식의 값은 x 의 값이다.

◆ $(x == y)$

- x 와 y 가 같으면 1, 다르면 0이 수식의 값이 된다.

◆ $\text{if}(x == y)$ 를 $\text{if}(x = y)$ 로 잘못 쓰지 않도록 주의!



논리 연산자 (logical operator)

- ◆ 여러 개의 조건을 조합하여 참과 거짓을 따지는 연산자
- ◆ 결과값은 참(1) 아니면 거짓(0)

연산자 기호	사용 예	의미
&&	x && y	AND 연산, x와 y가 모두 참이면 참, 그렇지 않으면 거짓
	x y	OR 연산, x나 y중에서 하나만 참이면 참, 모두 거짓이면 거짓
!	!x	NOT 연산, x가 참이면 거짓, x가 거짓이면 참

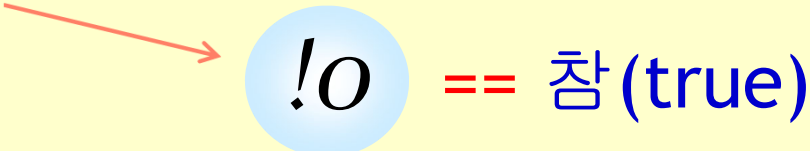
x	y	x && y	x y
참	참	참	참
참	거짓	거짓	참
거짓	참	거짓	참
거짓	거짓	거짓	거짓



참과 거짓의 표현 방법

- ◆ 관계 수식이나 논리 수식이 만약 참이면 1이 생성되고 거짓이면 0이 생성된다.
- ◆ 피연산자의 참, 거짓을 가릴 때에는 0이면 거짓이고, 0이 아니면 참으로 판단한다.
- ◆ 음수도 참으로 판단한다.
- ◆ (예) NOT 연산자를 적용하는 경우

```
!0      // 식의 값은 1
!3      // 식의 값은 0
!!3     // 식의 값은 1
!-3     // 식의 값은 0
```



!0 == 참(true)

논리 연산자의 예

◆ "x는 1, 2, 3중의 하나인가"

- $(x == 1) \parallel (x == 2) \parallel (x == 3)$

◆ "x가 60이상 100미만이다."

- $(x \geq 60) \&\& (x < 100)$

◆ "x가 0도 아니고 1도 아니다."

- $(x \neq 0) \&\& (x \neq 1)$ // $x \neq 0$ 이고 $x \neq 1$ 이다.



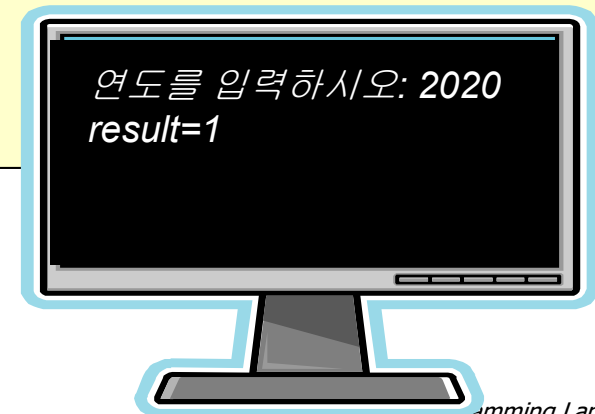
실습: 윤년 (leap year) 확인

```
// 윤년 확인 프로그램
#include <stdio.h>

int main(void)
{
    int year, result;

    printf("연도를 입력하시오: ");
    scanf("%d", &year);
    result = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0);
    printf("result=%d ", result);

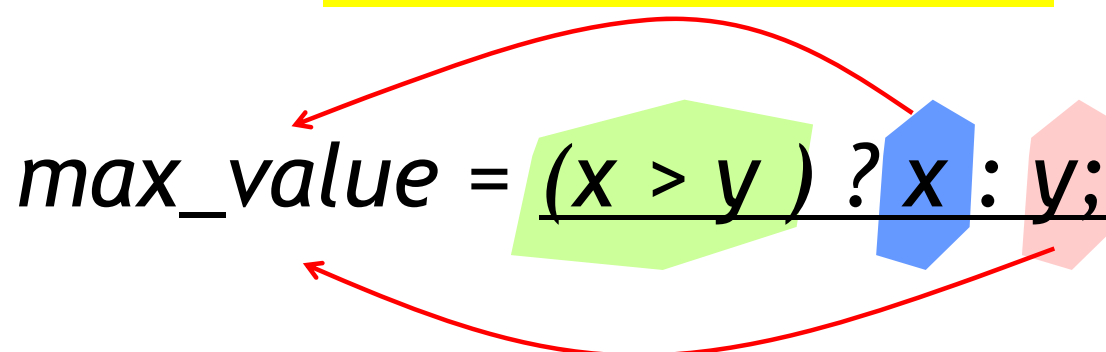
    return 0;
}
```



조건 연산자 (conditional operator)

$x > y$ 가 참이면 x 가 수식의 값이 된다.

$max_value = (x > y) ? x : y;$



$x > y$ 가 거짓이면 y 가 수식의 값이 된다.

```
absolute_value = (x > 0) ? x: -x; // 절대값 계산
max_value = (x > y) ? x: y;      // 최대값 계산
min_value = (x < y) ? x: y;      // 최소값 계산
(age > 20) ? printf("성인\n"): printf("청소년\n");
```

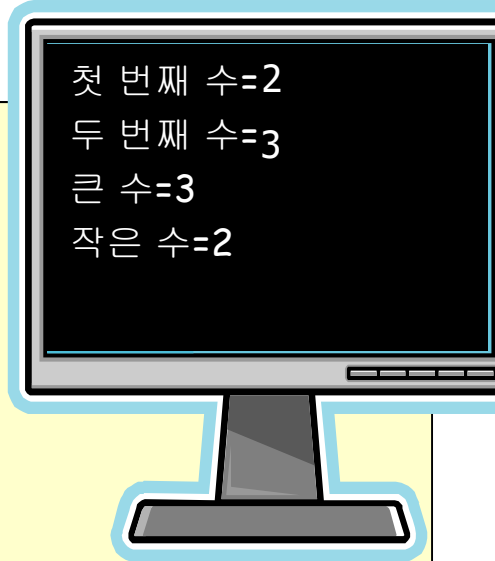


예제

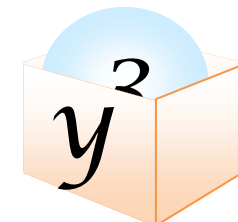
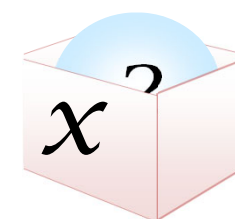
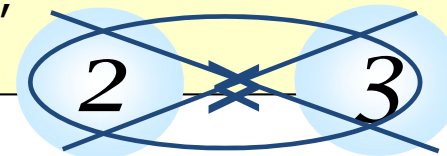
```
#include <stdio.h>
int main(void)
{
    int x,y;

    printf("첫 번째 수=");
    scanf("%d", &x);
    printf("두 번째 수=");
    scanf("%d", &y);

    → printf("큰 수=%d \n", (x > y) ? x : y);
    printf("작은 수=%d \n", (x < y) ? x : y);
}
```



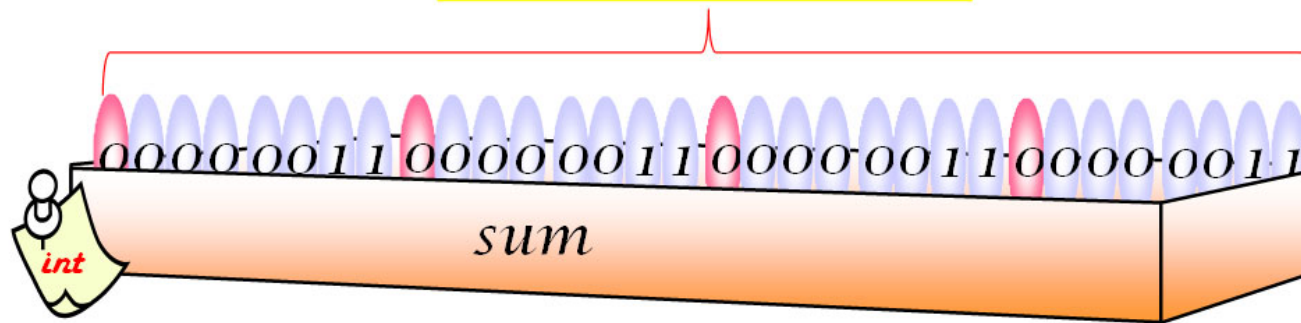
첫 번째 수=2
두 번째 수=3
큰 수=3
작은 수=2



비트 연산자 (bit operator)

연산자	연산자의 의미	설명
&	비트 AND	두개의 피연산자의 해당 비트가 모두 1이면 1, 아니면 0
	비트 OR	두개의 피연산자의 해당 비트중 하나만 1이면 1, 아니면 0
^	비트 XOR	두개의 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 연산 및 예제

Bitwise AND
$0 \& 0 \Rightarrow 0$
$1 \& 0 \Rightarrow 0$
$0 \& 1 \Rightarrow 0$
$1 \& 1 \Rightarrow 1$

Bitwise OR
$0 \mid 0 \Rightarrow 0$
$1 \mid 0 \Rightarrow 1$
$0 \mid 1 \Rightarrow 1$
$1 \mid 1 \Rightarrow 1$

Bitwise XOR
$0 \wedge 0 \Rightarrow 0$
$1 \wedge 0 \Rightarrow 1$
$0 \wedge 1 \Rightarrow 1$
$1 \wedge 1 \Rightarrow 0$

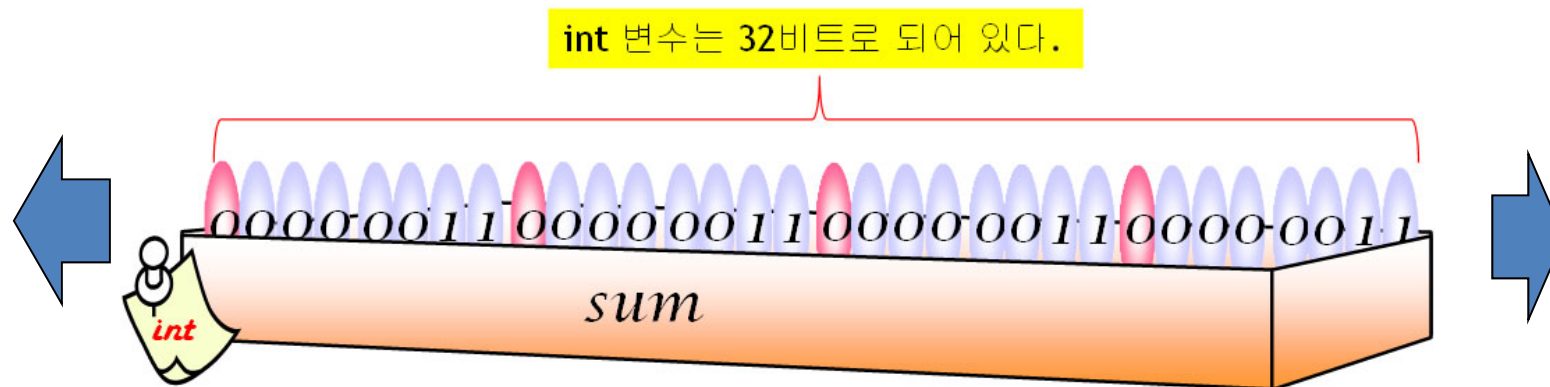
Bitwise NOT
$!0 \Rightarrow 1$
$!1 \Rightarrow 0$

```
Bitwise AND ( 00000111 & 10100011 ) => 00000011
Bitwise OR  ( 00000111 | 10100011 ) => 10100111
Bitwise XOR  ( 00000111 ^ 10100011 ) => 10100100
Bitwise Shift Left (00000111 << 3) => 00111000
Bitwise Shift Right (00000111 >> 3) => 00000000
Bitwise Shift Right (10100011 >> 3) => 11110100
계속하려면 아무 키나 누르십시오 . . .
```



비트 단위 자리 이동 (bitwise shift) 연산자

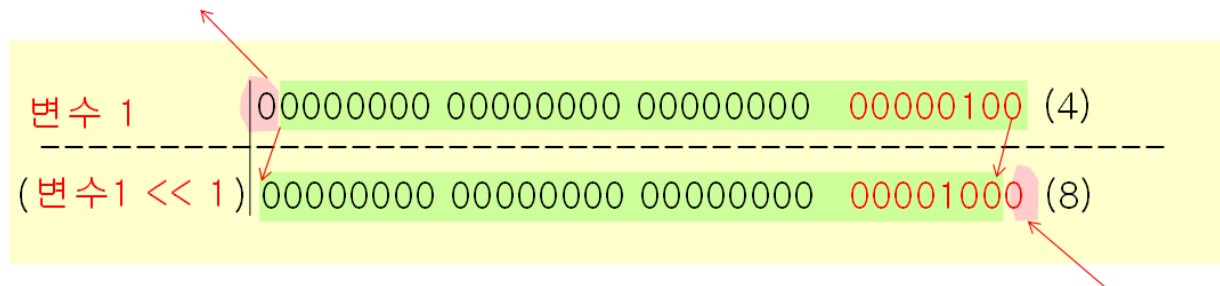
연산자	기호	설명
왼쪽 비트 자리 이동	<<	$x \ll y$ x의 비트들을 y 칸만큼 왼쪽으로 이동
오른쪽 비트 자리 이동	>>	$x \gg y$ x의 비트들을 y 칸만큼 오른쪽으로 이동



<< 연산자, >> 연산자

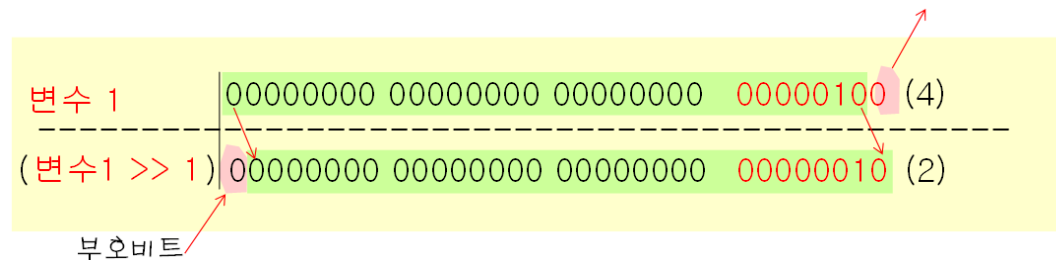
◆ Bitwise Shift Left

- 비트를 왼쪽으로 이동; 오른쪽에서 0 비트가 추가됨
- 값은 2배가 된다.



◆ Bitwise Shift Right

- 비트를 오른쪽으로 이동; 오른쪽에서 비트가 밀려남
- 값은 1/2배가 된다.
- 왼쪽 비트는 부호 비트가 반복됨



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;
}
```



우선 순위 (precedence)

◆ 어떤 연산자를 먼저 계산할 것인지에 대한 규칙

$$x + y * z$$

Diagram illustrating operator precedence for the expression $x + y * z$. The multiplication operation $y * z$ is performed first (labeled 1), followed by the addition operation $x + (y * z)$ (labeled 2).

$$(x + y) * z$$

Diagram illustrating operator precedence for the expression $(x + y) * z$. The addition operation $x + y$ is performed first (labeled 1), followed by the multiplication operation $(x + y) * z$ (labeled 2).

$$y = a \% b / c + d * (e - f);$$

Diagram illustrating operator precedence for the expression $y = a \% b / c + d * (e - f);$. The operations are performed in the following order: 1. $(e - f)$, 2. $a \% b$, 3. $(a \% b) / c$, 4. $d * (e - f)$, 5. $((a \% b) / c) + (d * (e - f))$, and finally 6. $y = ((a \% b) / c) + (d * (e - f))$.



우선 순위 (precedence)

우선 순위	연산자	결합 규칙
1	() [] -> . ++(후위) --(후위)	->(좌에서 우)
2	sizeof &(주소) ++(전위) --(전위) ~ ! *(역참조) +(부호) -(부호), 형 변환	<-(우에서 좌)
3	*(곱셈) / %	->(좌에서 우)
4	+(덧셈) -(뺄셈)	->(좌에서 우)
5	<< >>	->(좌에서 우)
6	< <= >= >	->(좌에서 우)
7	== !=	->(좌에서 우)
8	&(비트연산)	->(좌에서 우)
9	^	->(좌에서 우)
10		->(좌에서 우)
11	&&	->(좌에서 우)
12		->(좌에서 우)
13	?(삼항)	<-(우에서 좌)
14	= += *= /= %= &= ^= = <<= >>=	<-(우에서 좌)
15	,(coma)	->(좌에서 우)



연산자 우선 순위 (precedence)의 일반적인 지침

- ◆ 콤마 < 대입 < 논리 < 관계 < 산술 < 단항
- ◆ 괄호 연산자는 가장 우선순위가 높다.
- ◆ 모든 단 항 연산자들은 이항 연산자들보다 우선순위가 높다.
- ◆ 콤마 연산자를 제외하고는 대입 연산자가 가장 우선순위가 낮다.
- ◆ 연산자들의 우선 순위가 생각나지 않으면 괄호를 이용
 - $(x \leq 10) \ \&\& \ (y \geq 20)$
- ◆ 관계 연산자나 논리 연산자는 산술 연산자보다 우선순위가 낮다.
 - $x + 2 == y + 3$
// $(x + 2) == (y + 3)$ 으로 소스코드를 작성하는 것을 권장함



Homework 2

Homework 2

- 2.1 표준입력장치로 부터 초단위로 입력된 값을 시, 분, 초로 환산하는 프로그램의 **pseudo code**와 **C program**을 작성하라.
(예: 4255 sec => 1 시간 10 분 55 초)
- 2.2 C 프로그램을 디버깅할 때 **break point** 설정 방법과 **step-by-step tracing**하면서 중간 결과 값을 확인하는 방법에 대하여 설명하라.
- 2.3 1 ~ 32,765 범위의 정수 (**integer**) 값을 입력 받아, 만의 단위, 천의 단위, 백의 단위, 십의 단위, 일의 단위 값을 구분한 후, 차례로 출력하는 알고리즘의 **pseudo code**를 작성하고, 이 알고리즘에 대한 **C 프로그램**을 작성하라. 가장 높은 자리 수 이상의 값을 0으로 출력하지 말 것.
(예: 12345: 1만 2천 3백 4십 5
123: 1백 2십 5)
- 2.4 부동소수점 자료형의 실수 데이터 (**double type**) 2개를 입력 받아, 이들의 덧셈, 뺄셈, 곱셈, 나눗셈의 결과를 차례로 출력하는 프로그램의 **pseudo code**를 작성하고, **C 프로그램**을 작성하라. **Double** 실수 데이터의 출력은 소수점 이하 2자리까지 출력하게 할 것.

