

---

# 컴퓨터 프로그래밍 개론

## 데이터 표현과 연산

# 목차

---

- I. 변수
- II. 상수
- III. 연산자
- IV. 자료형
- V. 형 변환

# 변수의 개요

## ▶ 변수(variable)

- ▶ 프로그래밍에서 숫자나 문자 등의 데이터를 임시로 저장하기 위한 공간

- ▶ 물리적으로 메모리에 할당되는 저장 공간

- ▷ 변수를 선언하여 메모리에 저장 공간 확보

- ▷ 데이터를 메모리에 저장해 놓으면 필요할 때 마다 접근해서 사용 가능

- ▶ 변수를 선언하는 방법

자료형 변수명;

자료형 변수명 = 저장할 데이터;

- ▶ 변수의 종류

- ▷ 변수의 타입(형태)를 자료형(data type)이라고 하며 기본적으로 정수형과 실수형으로 구분

- ▷ 정수형 변수: char형(문자형), short형, int형, long형

- ▷ 실수형 변수: float형, double형, long double형

# 변수의 개요

## ▶ 변수의 선언방법

### ▶ 자료형 변수이름;

```
#include<stdio.h>
int main(void){
    int a;           // 정수형 변수 선언
    float b = 1.5;   // 실수형 변수 선언
    int c, d;        // 동일한 타입의 여러 개의 변수를 한꺼번에 선언

    return 0;
}
```

# 변수의 개요

## ▶ 변수의 사용

### ▶ 변수에 저장한 데이터는 변경가능

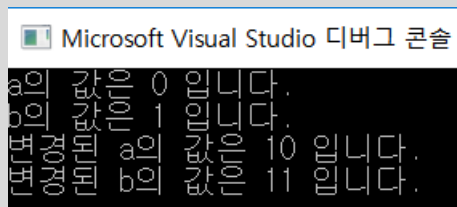
#### ▶ 변수 == 변하는 수

```
#include<stdio.h>
int main(void){
    int a = 0;
    int b = 1;

    printf("a의 값은 %d 입니다. \n", a);
    printf("b의 값은 %d 입니다. \n", b);

    a = a + 10;      //a += 10;
    b = b + 10;      //b += 10;

    printf("변경된 a의 값은 %d 입니다. \n", a);
    printf("변경된 b의 값은 %d 입니다. \n", b);
    return 0;
}
```



# 변수의 개요

## ▶ 변수 선언 시 주의할 점

- ▶ 한 구역{}안에서 변수의 이름이 동일하게 중복될 수 없음
  - ▶ 자료형이 다르더라도 변수이름은 같을 수 없음

```
#include<stdio.h>
int main(void){
    int a;
    float a;          //error

    //a = 0 어떤 a에 저장??

    return 0;
}
```

# 변수의 개요

## ▶ 변수 선언 시 주의할 점

▶ 변수는 소문자로 작성

▶ 특수 기호와 공백 문자는 변수의 이름에 사용할 수 없으며 변수의 첫 글자로 숫자를 사용할 수 없음

▶ 그러나 밑줄(언더바 : \_)나 달러표시\$는 사용 가능

int Apple;	int ?apple;	특수문 자 ?사 용
int total;	int to tal;	
int result2;	int 2result;	

▶ 대소문자를 구분

▶ student != Student

		설명
int Apple;	int apple;	
int TOTAL;	int total;	
int result;	int result;	

# 변수의 개요

## ▶ 변수 선언 시 주의할 점

- ▶ 프로그래밍에서 사용되는 키워드(예약어)를 변수 이름에 사용 불가

int int;	
int long;	
int short;	

- ▶ ANSI(American National Standards Institute)에 키워드들

- ▶ C에서 사용되는 예약어

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



# 변수의 개요

## ▶ 변수 선언 시 주의할 점

### ▶ 변수의 선언은 코드의 앞쪽에 작성

▷ 컴파일러에 따라 오류가 발생할 수도 있음

▷ 소스 코드 분석 시 가독성이 떨어짐

```
#include<stdio.h>
int main(void){
    int a;
    int b;

    a = 1;
    b = 2;

    int c; // 코드의 중간에 변수를 선언하는 것은 권장하지 않음
    ...
}
```

▶ 컴파일러의 버전에 따라 변수 선언 위치의 제한 여부가 달라지기는 하나, 가능하면 사용할 모든 변수를 블록의 시작부분에 선언

# 변수의 개요

## ▶ 변수 선언 시 주의할 점

- ▶ 변수의 이름을 프로그램 내에서 의미 있게 작성
- ▶ 의미 있는 변수명으로 선언해야 코드의 가독성이 높아짐
- ▶ 예제
  - ▶ 창고에 있는 과일박스의 총 개수를 알려주는(출력하는) 프로그램을 작성하시오.

```
#include<stdio.h>
int main(void){
    int a = 30;    //사과박스의 개수
    int b = 20;    //포도박스의 개수
    int c;         //총 박스의 개수

    c = a + b;
    printf("총 %d 박스가 있습니다 \n", c);

    return 0;
}
```

```
#include<stdio.h>
int main(void){
    int appleBox = 30;
    int grapeBox = 20;
    int total;

    total = appleBox + grapeBox;
    printf("총 %d 박스가 있습니다 \n", total);

    return 0;
}
```

# 변수의 개요

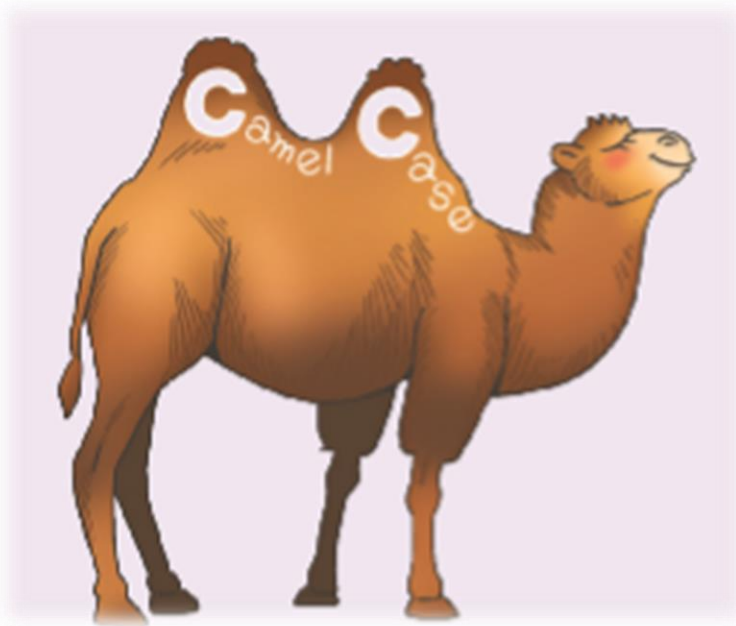
## ▶ 변수 선언 시 주의할 점

### ▶ 카멜 표기법(낙타체 : camel case)

▶ 변수의 이름을 여러 단어의 조합으로 작성할 경우에 단어 의 첫 글자는 대문자로 적는 방법

▶ 그러나 변수임을 의미하도록 첫 단어의 첫 글자는 소문자로 표기

▶ myNewCar, appleBox, saveGame, loadGame...



# 변수의 개요

## ▶ 변수 선언 시 주의할 점

▶ 변수를 초기화하지 않으면 프로그램 실행 과정에서 쓰레기 값으로 처리됨

▶ '0'이 출력되지 않을까? ➔ 0도 엄연한 데이터

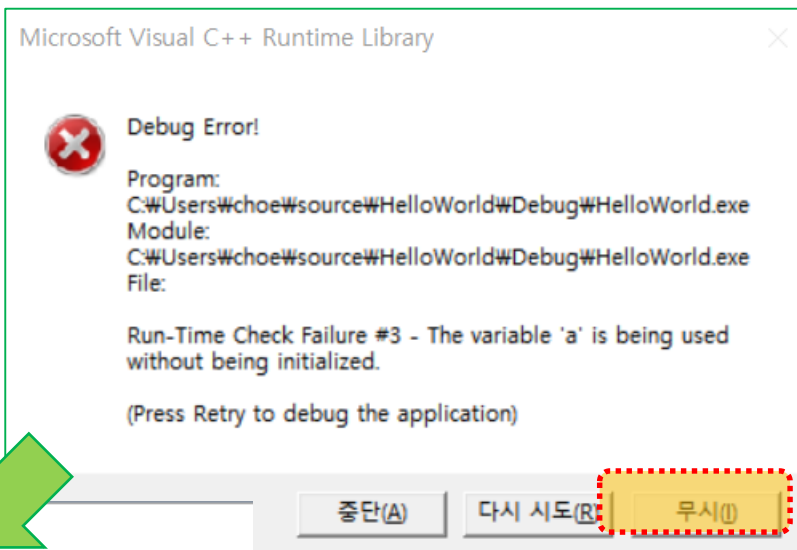
▶ 쓰레기 값 출력 예제

▶ VS2019에서는 보안이 강화되어 초기화되지 않은 변수가 존재할 경우 프로그램이 실행되지 않음

```
#include<stdio.h>

int main(void)
{
    int a;
    printf("%d\n", a);

    return 0;
}
```



# 변수의 개요

## ▶ 변수 선언 시 주의할 점

### ▶ 쓰레기 값

#### ▷ 변수 선언 시 할당된 메모리에 이미 저장되어 있는 값

▷ 이전 프로그램에서 사용하던 데이터

#### ▷ 어떤 값이 들어가 있을지 모름

▷ 이전 프로그램이 종료되면서 메모리의 해당 주소에 어떠한 값을 남겨져 있을지 예상할 수 없음

▷ 어떤 값이 들어있든 현재의 프로그램에서는 의미가 없는 값이므로 쓰레기 값(garbage)이라고 부름

### ▶ 초기화

#### ▷ 프로그램의 동작 과정에서 쓰레기 값을 사용하지 않도록 변수를 선언한 후 최초로 데이터를 저장(대입)하는 것

#### ▷ 초기화는 변수를 사용하기 전에 반드시 선행되어야 하는 단계

#### ▷ 선언과 동시에 초기화, 또는 선언 후 나중에 초기화 가능

```
int a = 10  
int b;  
b = 20
```

# 상수의 개요

## ▶ 상수(constant)

- ▶ 어떤 상황에서도 변하지 않는 값을 의미
- ▶ 프로그램에서 모든 데이터는 변수 또는 상수의 형태로 사용
- ▶ 프로그래밍에서 상수는 리터럴(literal) 상수와 심볼릭(symbolic) 상수로 구분
  - ▷ 리터럴 상수는 일반적으로 사용하는 값
    - ▷ a, b, c, 1, 2, 3, D, E, F, ...
  - ▷ 심볼릭 상수는 변수와 선언하는 방법이 유사하며 한번 값이 결정되면 절대로 변경할 수 없음
    - ▷ 변수와 달리 상수는 한번 초기화한 데이터는 이후에 변경할 수 없음

# 상수의 개요

## ▶ 리터럴 상수

▶ 글자 그대로의 의미가 있어서 이름이 없는 상수

▶ a, b, c, 1, 2, 3, D, E, F, ...

▶ 정수형 상수, 실수형 상수, 문자 상수, 문자열 상수

▶ 상수를 구분하기 위하여 8진수는 0을 16진수는 0x를 숫자 앞에 붙여서 10진수와 다른 상수임을 표시

```
#include<stdio.h>
int main(void){
    char c = 'a';
    printf("변수 c에는 문자 %c가 저장되어 있습니다.\n", c);
    printf("10진수 정수형 상수 %d + %d = %d 입니다. \n", 10, 20, 10 + 20); //10진수
    printf("16진수 정수형 상수 %x + %x = %x 입니다. \n", 0x10, 0x20, 0x10 + 0x20); //16진수
    printf(" 8진수 정수형 상수 %o + %o = %o 입니다. \n", 010, 020, 010 + 020); //8진수
    return 0;
}
```

변수 c에는 문자 a가 저장되어 있습니다.  
10진수 정수형 상수 10 + 20 = 30 입니다.  
16진수 정수형 상수 10 + 20 = 30 입니다.  
8진수 정수형 상수 10 + 20 = 30 입니다.

# 상수의 개요

## ▶ 리터럴 상수

### ▶ 문자 상수를 출력하기 위하여 %c 서식 문자사용

```
#include<stdio.h>
int main(void)
{
    printf("문자 상수 %c %c %c 입니다.\n", 'a', 'b', 'c'); //문자 상수 a b c 입니다.
    printf("문자 상수 %c %c %c 입니다.\n", '!', '@', '#'); //문자 상수 ! @ # 입니다.
    return 0;
}
```

문자 상수 a b c 입니다.  
문자 상수 ! @ # 입니다.

### ▶ %d를 사용하여 문자를 출력하면 문자의 아스키 코드 값이 출력

```
#include<stdio.h>
int main(void)
{
    printf("문자 상수 %d %d %d 입니다.\n", 'a', 'b', 'c'); //문자 상수 97 98 99 입니다.
    printf("문자 상수 %d %d %d 입니다.\n", '!', '@', '#'); //문자 상수 33 64 35 입니다.
    return 0;
}
```

문자 상수 97 98 99 입니다.  
문자 상수 33 64 35 입니다.



# 상수의 개요

## ▶ 리터럴 상수

- ▶ 문자열 상수는 %s 서식문자를 사용하여 출력

```
#include<stdio.h>
int main(void)
{
    printf("문자열 상수는 %s 입니다. \n", "A");           //문자열 상수는 A 입니다.
    printf("문자열 상수는 %s 입니다. \n", "10+10");       //문자열 상수는 10+10 입니다.
    printf("문자열 상수는 %s 입니다. \n", "Hi, everyone"); //문자열 상수는 Hi, everyone 입니다.
    return 0;
}
```

문자열 상수는 A 입니다.  
문자열 상수는 10+10 입니다.  
문자열 상수는 Hi, everyone 입니다.

## ▶ 상수가 컴파일 된 후의 비트 형태

- ▶ 입력한 문자들이 모두 아스키 코드 값으로 저장
- ▶ 예를 들어 "10+10" 코드는 '1', '0', '+', '1', '0', ';' 등의 토큰(token)으로 분리되어 저장

# 상수의 개요

## ▶ 심볼릭 상수

- ▶ 변수처럼 선언하고 접근이 가능하지만 한번 초기화되면 값을 변경할 수 없음
- ▶ 프로그래밍 내에서 상수임을 나타내기 위하여 대문자로 표기
- ▶ 코드 내부에서 값이 변하면 위험한 경우 주로 사용
  - ▷ 은행의 이자율, 원주율(자릿수) 등에 사용
- ▶ #define문을 사용하여 전처리기에서 선언
- ▶ const 키워드를 사용하여 코드 내에서 선언
  - ▷ const 사용 시 반드시 선언과 동시 초기화
  - ▷ 쓰레기 값 할당 방지

```
#include<stdio.h>
int main(void)
{
    const int NUM = 100;
    const double PI = 3.14;
    return 0;
}
```

```
#include <stdio.h>

#define PI 3.14
#define NUM 100
#define BUFFER_SIZE 200

int main(){
    printf("%lf \n", PI);
    printf("%d \n", NUM);
    printf("%d \n", BUFFER_SIZE);

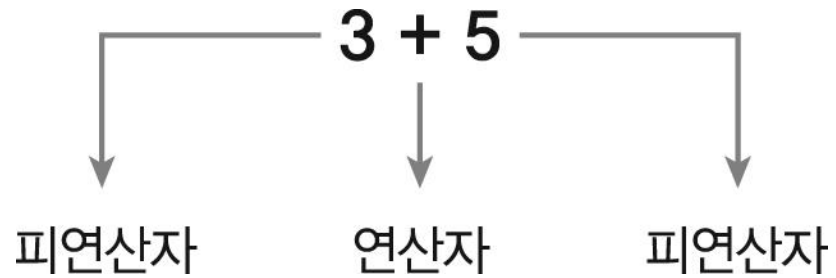
    return 0;
}
```

```
3.140000
100
200
```

# 연산자의 개요

## ▶ 연산자와 피연산자

- ▶ 수식(expression) : 피연산자들과 연산자의 조합
- ▶ 연산자(operator): 연산을 수행하는 기호
- ▶ 피연산자(operand) : 연산에 참여하는 변수나 상수 값



# 연산자의 개요

## ▶ 연산자의 종류

- ▶ 수식에서 순서에 상관없이 곱셈을 덧셈보다 먼저 계산하듯이 프로그래밍에서도 연산자의 우선순위가 존재

분 류	
	=
	+, -, *, /, %
	+=, -=, *=, /=, %=
	++, --
	>, <, ==, !=, >=, <=
	&&,   , !
	? :
	&, !, ^, ~
	>>, <<

# 연산자의 개요

## ▶ 대입 연산자

▶ =

## ▶ 데이터를 저장하는 연산자

▶ 대입 연산자의 오른쪽에 있는 데이터나 계산 결과를 왼쪽의 저장공간으로 넘겨줌(저장)

```
#include<stdio.h>
int main(void){
    int i = 3; //i에 3을 저장한다는 의미
    return 0;
}
```

```
#include<stdio.h>
int main(void) {
    int i = 0, j = 0, k = 0;
    printf("i = %d, j = %d, k = %d \n", i, j, k);    //i = 0, j = 0, k = 0
    i = 1;
    j = 5;
    k = 7;
    printf("i = %d, j = %d, k = %d\n", i, j, k);    //i = 1, j = 5, k = 7
    return 0;
}
```

i = 0, j = 0, k = 0
i = 1, j = 5, k = 7

# 연산자의 개요

## ▶ 산술 연산자

▶ 덧셈(+), 뺄셈(-), 곱셈(\*), 나눗셈(/), 나머지(%)

산술 연 자	예	설명
+ (덧셈 연 자)	$a = 6 + 2$	피연 자 6과 피연 자 2의 덧셈 연
- (뺄셈 연 자)	$a = 6 - 2$	피연 자 6과 피연 자 2의 뺄셈 연
* (곱셈 연 자)	$a = 6 * 2$	피연 자 6과 피연 자 2의 곱셈 연
/ (나눗셈 연 자)	$a = 6 / 2$	피연 자 6과 피연 자 2의 나눗셈 연
% (나머지 연 자)	$a = 6 \% 2$	피연 자 6과 피연 자 2를 나눈 나머지 연

```
#include<stdio.h>
int main(void){
    int a, b;
    a = 6;
    b = 2;
    printf("덧셈 연 결과: %d \n", a + b); //덧셈 연 결과: 8
    printf("뺄셈 연 결과: %d \n", a - b); //뺄셈 연 결과: 4
    printf("곱셈 연 결과: %d \n", a * b); //곱셈 연 결과: 12
    printf("나눗셈 연 결과: %d \n", a / b); //나눗셈 연 결과: 3
    printf("나머지 연 결과: %d \n", a % b); //나머지 연 결과: 0
    return 0;
}
```

덧셈 연	결과: 8
뺄셈 연	결과: 4
곱셈 연	결과: 12
나눗셈 연	결과: 3
나머지 연	결과: 0

# 연산자의 개요

## ▶ 복합 대입 연산자

### ▶ 산술 연산자와 대입 연산자를 하나로 나타내는 기호

복합대입 연산자	같은표현	설명
<code>a = a + b</code>	<code>a += b</code>	<code>a + b</code> 를 먼저 수행한 후에 <code>a</code> 에 값을 저장
<code>a = a - b</code>	<code>a -= b</code>	<code>a - b</code> 를 먼저 수행한 후에 <code>a</code> 에 값을 저장
<code>a = a * b</code>	<code>a *= b</code>	<code>a * b</code> 를 먼저 수행한 후에 <code>a</code> 에 값을 저장
<code>a = a / b</code>	<code>a /= b</code>	<code>a / b</code> 를 먼저 수행한 후에 <code>a</code> 에 값을 저장
<code>a = a % b</code>	<code>a %= b</code>	<code>a % b</code> 를 먼저 수행한 후에 <code>a</code> 에 값을 저장

```
#include<stdio.h>
int main(void) {
    int num1 = 1, num2 = 2, num3 = 3, num4 = 4, num5 = 5;
    num1 += num2;    //num1 = num1 + num2;
    num2 -= 2;       //num2 = num2 - 2;
    num3 *= 2;       //num3 = num3 * 2;
    num4 /= 2;       //num4 = num4 / 2;
    num5 %= 2;       //num5 = num5 % 2;
    printf("%d, %d, %d, %d, %d \n", num1, num2, num3, num4, num5);    //3, 0, 6, 2, 1

    return 0;
}
```

3, 0, 6, 2, 1

# 연산자의 개요

## 증감 연산자

▶ ++, -- 기호를 이용하는 연산자 (1증가 또는 1감소 시키는 연산자)

▶ 기호의 위치에 따라 전위와 후위로 구분되며 실행 시점이 달라짐

증감 연 자	실행순 서
++a	선증 가, 후 연 (먼 저증 가화 그 다음 연 )
a++	선 연 , 후증 가 (먼 저 연 화 그 다음증 가)
--a	선감 소, 후 연 (먼 저감 소화 그 다음 연 )
a--	선 연 , 후감 소 (먼 저 연 화 그 다음감 소)

```
#include<stdio.h>
int main(void){
    int num1=10, num2=10;
    int a, b;
    a = ++num1;           // 전위 방식, 선증 가 후 연
    printf("%d, %d \n", a, num1); // 결과는 11, 11
    b = num2++;           // 후위 방식, 선 연 후증 가
    printf("%d, %d \n", b, num2); // 결과는 10, 11
    return 0;
}
```

11, 11  
10, 11

```
#include<stdio.h>
int main(void) {
    int num1 = 10, num2 = 10;
    printf("%d \n", num2);
    num2 = num1++;
    printf("%d \n", num2);
    num2 = ++num1;
    printf("%d \n", num2);
    num2 = --num1;
    printf("%d \n", num2);
    num2 = num1--;
    printf("%d \n", num2);
    return 0;
}
```

10  
10  
12  
11  
11



# 연산자의 개요

## ▶ 관계 연산자

▶ ‘비교연산자’라고도 함

▶ 관계를 비교하여 참(True)과 거짓(False)으로 결론짓는 연산자

관계 연 산 자	예	설명	결과
>	$a > b$	a가 b보다 클 지를비 교	1참 ), 0거 짓 )
<	$a < b$	a가 b보다작 을지를비 교	1참 ), 0거 짓 )
>=	$a \geq b$	a가 b보다크 거 나 같을지를비 교	1참 ), 0거 짓 )
<=	$a \leq b$	a가 b보다작 거 나 같을지를비 교	1참 ), 0거 짓 )
==	$a == b$	a가 b보다 같을지를비 교	1참 ), 0거 짓 )
!=	$a != b$	a가 b보다 같지 않을지를비 교	1참 ), 0거 짓 )

# 연산자의 개요

## ▶ 관계 연산자

### ▶ 관계를 비교하여 참(True)과 거짓(False)으로 결론짓는 연산자

```
#include<stdio.h>
int main(void) {
    int num1 = 2, num2 = 4;
    int result1, result2, result3, result4;

    result1 = (num1 > num2);
    result2 = (num1 <= num2);
    result3 = (num1 == num2);
    result4 = (num1 != num2);

    printf("result1에 저장된 값 %d \n", result1);    // 0(거짓)
    printf("result2에 저장된 값 %d \n", result2);    // 1(참)
    printf("result3에 저장된 값 %d \n", result3);    // 0(거짓)
    printf("result4에 저장된 값 %d \n", result4);    // 1(참)

    return 0;
}
```

result1에	저장된	값	0
result2에	저장된	값	1
result3에	저장된	값	0
result4에	저장된	값	1

# 연산자의 개요

## ▶ 논리 연산자

▶ 관계연산자의 결과에 따라 연산이 되는 연산자

▶ && : AND 연산자 (논리곱)

▶ || : OR 연산자 (논리합)

▶ ! : NOT 연산자 (논리 부정)

피연 자	연 자	피연 자	결과
0	&&	0	0(거짓)
0	&&	1	0(거짓)
1	&&	0	0(거짓)
1	&&	1	1참)

피연 자	연 자	피연 자	결과
0		0	0(거짓)
0		1	1참)
1		0	1참)
1		1	1참)

연 자	피연 자	결과
!	0	1참)
!	1	0(거짓)

# 연산자의 개요

## ▶ 논리 연산자

- ▶ && : 모든 조건식의 결과가 참이어야 AND연산자의 결과가 참으로 결정 / 하나라도 거짓이면 결과는 거짓
- ▶ || : 모든 조건식의 결과가 거짓이어야 OR연산자의 결과가 거짓으로 결정 / 하나라도 참이면 결과는 참
- ▶ ! : 조건식의 결과가 거짓이면 NOT연산자의 결과는 참 / 참이면 거짓

```
#include<stdio.h>
int main(void){
    int num1=2, num2=3, num3=5;
    int result1, result2, result3;

    result1 = (num1>0) && (num2<10);
    result2 = (num2<=2) || (num3>5);
    result3 = !num3;

    printf("result1에 저장된 값 %d \n", result1); // 1(참)
    printf("result2에 저장된 값 %d \n", result2); // 0(거짓)
    printf("result3에 저장된 값 %d \n", result3); // 0(거짓)

    return 0;
}
```

result1에	저장된	값	1
result2에	저장된	값	0
result3에	저장된	값	0

# 연산자의 개요

## ▶ 조건 연산자

▶ ‘?’ 와 ‘:’ 로 이루어진 연산자

▶ 작성법

▶ 조건식 ? 식1 : 식2

▶ 조건식의 판단 결과에 따라 참이면 식1을, 거짓이면 식2를 수행

```
#include<stdio.h>
int main(void)
{
    int num1 = 2, num2 = 3;
    int result;

    result = (num1 > num2) ? num1 : num2;
    printf("result에 저장된 값 %d \n", result);

    return 0;
}
```

result에 저장된 값 3

# 연산자의 개요

## ▶ 비트(bit)

- ▶ binary digit의 약자
- ▶ 컴퓨팅 세계의 표현법
  - ▶ 컴퓨터는 모든 데이터를 2진수로 표현
- ▶ 2진수 값 하나(0 또는 1)를 저장할 수 있는 최소 메모리 공간
  - ▶ 논리적으로는 참(true) 또는 거짓(false)을 나타냄

1비트	2비트	3비트	...	n비트
$2^1=2$ 개	$2^2=4$ 개	$2^3=8$ 개	...	$2^n$ 개

## ▶ 바이트(byte)

- ▶ 8개의 비트로 구성된 저장 단위
- ▶ 1byte = 8bit
- ▶ 실질적 의미는 ASCII 문자 하나의 크기를 뜻함



# 연산자의 개요

## ▶ 비트 연산자

### ▶ 데이터를 비트 단위로 처리하는 연산자

비트 연 자	연 식	설명
&	$a \& b$	비트단 위 AND 연
	$a   b$	비트단 위 OR 연
^	$a \wedge b$	비트단 위 XOR 연
~	$\sim a$	비트단 위 NOT 연
<<	$a << 3$	
>>	$a >> 1$	

# 연산자의 개요

## ▶ 비트 연산자

### ▶ & 연산자

▶ 비트 단위로 AND 연산을 수행

▶ 두 개의 비트가 모두 1일 때 1을 반환

피연 자	연 자	피연 자	결과
0	&	0	0
0	&	1	0
1	&	0	0
1	&	1	1

```
#include<stdio.h>
int main(void){
    int num1=20, num2=16;
    int result;

    result = num1 & num2;
```

&

0000	0000	0000	0000	0000	0000	0001	0100	(20)
0000	0000	0000	0000	0000	0000	0001	0000	(16)
0000	0000	0000	0000	0000	0000	0001	0000	(16)



# 연산자의 개요

## ▶ 비트 연산자

### ▶ | 연산자

▷ 비트 단위로 OR 연산을 수행

▷ 두 개의 비트 중의 하나가 1일 때 1을 반환

피연 자	연 자	피연 자	결과
0		0	0
0		1	1
1		0	1
1		1	1

```
#include<stdio.h>
int main(void){
    int num1=20, num2=16;
    int result;

    result = num1 | num2;
```

	0000	0000	0000	0000	0000	0000	0001	0100	(20)
	0000	0000	0000	0000	0000	0000	0001	0000	(16)
	0000	0000	0000	0000	0000	0000	0001	0100	(20)

# 연산자의 개요

## ▶ 비트 연산자

### ▶ ^ 연산자

▶ 비트 단위로 XOR 연산을 수행

▶ 두 개의 비트가 서로 같지 않을 경우 1을 반환

피연 자	연 자	피연 자	결과
0	^	0	0
0	^	1	1
1	^	0	1
1	^	1	0

```
#include<stdio.h>
int main(void){
    int num1=20, num2=16;
    int result;

    result = num1 ^ num2;
```

^

0000	0000	0000	0000	0000	0000	0001	0100	(20)
0000	0000	0000	0000	0000	0000	0001	0000	(16)
0000	0000	0000	0000	0000	0000	0000	0100	(4)

# 연산자의 개요

## ▶ 비트 연산자

### ▶ ~ 연산자

▷ 비트 단위로 NOT 연산을 수행

▷ 보수 연산으로 비트를 반전 시킴

연 산 자	피연 산 자	결과
~	0	1
~	1	0

```
#include<stdio.h>
int main(void){
    int num1=20;
    int result1;

    result1 = ~num1;
```

~    0000 0000    0000 0000    0000 0000    0001 0100    (20)

     1111 1111    1111 1111    1111 1111    1110 1011    (-21)

# 연산자의 개요

## ▶ 비트 연산자

### ▶ 비트로 음수 표현

▷ 절대값의 보수로 변경 후 + 1

▷  $21 + (-21) = 0$

1의 보수 1의 더함	0000	0000	0000	0000	0000	0000	0001	0101	(21)
	1111	1111	1111	1111	1111	1111	1110	1010	
	0000	0000	0000	0000	0000	0000	0000	0001	
<hr/>									
	1111	1111	1111	1111	1111	1111	1110	1011	(-21)

을

# 연산자의 개요

## ▶ 비트 이동 연산자

### ▶ <<연산자 (왼쪽 시프트 연산자)

▷ 비트를 왼쪽으로 이동시키는 연산자

### ▶ >>연산자 (오른쪽 시프트 연산자)

▷ 비트를 오른쪽으로 이동시키는 연산자

```
#include<stdio.h>
int main(void){
    int num1=10;
    int result;

    result = num1 << 2;
```

# 연산자의 개요

## ▶ 비트 이동 연산자

### ▶ <<연산자 (왼쪽 시프트 연산자)

▷ 비트를 왼쪽으로 이동시키는 연산자

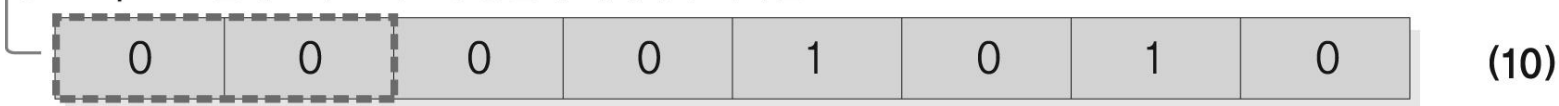
### ▶ >>연산자 (오른쪽 시프트 연산자)

▷ 비트를 오른쪽으로 이동시키는 연산자

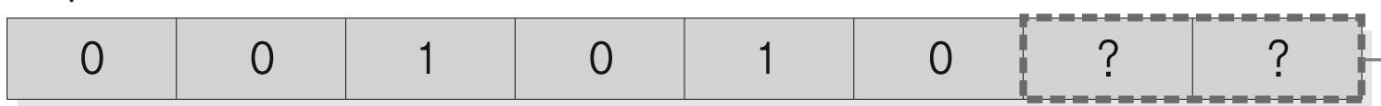
### ▶ 예제

▷ `result1 = num1 << 2;` ➔ 왼쪽으로 1비트 씩 이동할 때 마다 2씩 곱함

➔ Step 1 : 왼쪽 2비트 메모리 공간의 데이터가 사라짐



Step 2 : 오른쪽에 2비트 빈 메모리 공간이 생김



Step 3 : 빈 메모리 공간을 0으로 채움



# 연산자의 개요

## ▶ 비트 이동 연산자

### ▶ 오른쪽으로 1비트씩 이동할 때 마다 2씩 나누어짐

```
#include<stdio.h>
int main(void){
    int num = 10;

    int result;

    result = num >> 1;
```

Step 1 : 오른쪽 1비트 메모리 공간의 데이터가 사라짐



Step 2 : 왼쪽에 1비트 빈 메모리 공간이 생김



Step 3 : 빈 메모리 공간을 0으로 채움



# 연산자의 개요

## ▶ 비트 이동 연산자

### ▶ 오른쪽으로 1비트씩 이동할 때 마다 2씩 나누어짐

```
#include<stdio.h>
int main(void){
    int num = -10;

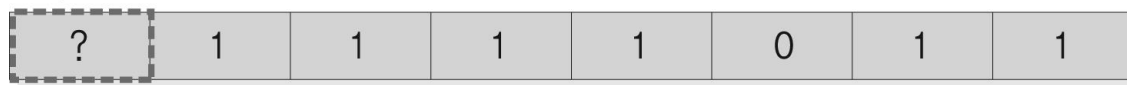
    int result;

    result = num >> 1;
```

Step 1 : 오른쪽 1비트 메모리 공간의 데이터가 사라짐



Step 2 : 왼쪽에 1비트 빈 메모리 공간이 생김



Step 3 : 빈 메모리 공간을 1로 채움





# 연산자의 개요

## ▶ 연산자 우선순위

▶ 암기하기 복잡하므로 수식을 작성하는 과정에서 애매한 경우에는 소괄호( )를 사용

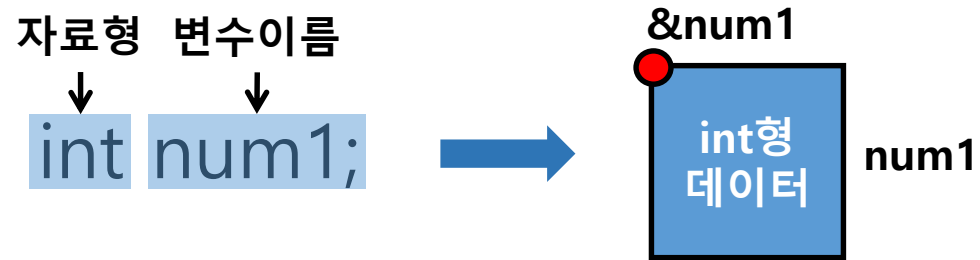
우선 위	연 자	연 방
1	() [] -> .	
2	~ ~ ++ -- + - * &	
3	* / %	
4	+ -	
5	<< >>	
6	< <= > >=	
7	== !=	
8	&	

우선 위	연 자	연 방
9	^	
10		
11	&&	
12		
13	?:	
14	= += -= *= /= %= &= ^= != <<= >>=	
15	,	

# 자료형의 개요

## ▶ 자료형

- ▶ 변수의 데이터 형식
- ▶ 데이터 타입을 이해해야 효율적이고 신뢰성 있는 프로그램 작성 가능



## ▶ 자료형의 종류

- ▶ 정수형: 정수를 표현하는 데이터 타입
- ▶ 실수형: 소수점이 포함된 값을 표현하는 데이터 타입

정수형				실수형		
char	short	int	long	float	double	long double
문자형						

# 자료형의 개요

## ▶ 자료형의 크기

- ▶ 저장되는 데이터의 크기에 따라 적절한 크기의 저장공간을 사용해야 함
- ▶ 프로그램 동작 과정에서는 다양한 데이터가 사용되므로 이에 따라 효율적인 변수 사용이 필요함
- ▶ C언어에서는 다양한 자료형을 사용할 수 있음
  - ▷ 초기의 컴퓨터는 낮은 사양이었기 때문에 적은 메모리 공간을 효율적으로 사용할 필요가 있었음
- ▶ 현재는 물리적인 메모리의 크기가 커지고 가격도 낮아졌지만 그 만큼 많은 양의 데이터가 생산됨
  - ▷ 프로그래밍 과정에서 효율적인 메모리 사용이 여전히 요구됨



# 자료형의 개요

## ▶ 자료형의 크기

### ▶ 기본 자료형의 메모리 크기

정수형			
char	short	int	long
1바이트	2바이트	4바이트	4바이트
문자형			

실수형		
float	double	long double
4바이트	8바이트	8바이트

### ▶ sizeof 연산자

#### ▶ 자료형의 크기를 구하는 연산자

#### ▶ 자료형에 할당되는 메모리의 크기를 구할 수 있음

용 법	예	설명
sizeof(자료 형)	printf( "%d", sizeof(int) );	자료 형의 메모리 크기를 출력
sizeof(변수)	int num1 = 3; printf( "%d", sizeof(num1) );	변수의 메모리 크기를 출력

# 자료형의 개요

## ▶ char 형

- ▶ 문자 데이터를 저장할 경우 사용
- ▶ 문자를 아스키 코드(ASCII code)값으로 처리
  - ▶ 프로그램에서 사용하는 모든 문자를 0~127 사이의 정수로 바꾸어 처리
    - ▶ char형 변수를 사용하면 문자를 가장 효과적으로 저장
- ▶ char형 데이터를 출력할 경우에 %c 사용시 문자로 출력
  - ▶ %d 사용시 아스키 코드 값 출력
    - ▶ 아스키 코드 값을 프로그램에 직접 사용할 수 있음
    - ▶ 예제 : 문자 'A'의 아스키 코드 값은 65

```
char ch1 = 'a';  
char ch2 = 65;  
  
printf("문자 %c의 아스키 코드 값 : %d \n", ch1, ch1);  
printf("아스키 코드 값이 : %d인 문자 : %c \n", ch2, ch2);
```

```
문자 a의 아스키 코드 값 : 97  
아스키 코드 값이 : 65인 문자 : A
```

```
-----  
Process exited after 0.7067 seconds with return value 0  
계속하려면 아무 키나 누르십시오 . . .
```

# 자료형의 개요

## ▶ ASCII(American Standard Code Information Interchange)

- ▶ 컴퓨터에서 문자를 숫자로 표현하기 위한 약속
  - ▷ 다른 구조의 시스템 간에 네트워크를 통하여 동일한 데이터를 주고 받을 수 있도록 서로 약속한 표준
- ▶ 컴퓨터는 모든 데이터를 숫자로 인식하기 때문에 문자도 숫자로 변경하여 처리해야 함
- ▶ 128개의 문자를 코드화
  - ▷ 영문 대소문자, 숫자, 특수문자, 제어문자 등이 포함(아스키 코드 표 참고)
- ▶ 'A'라는 문자를 숫자로 표현할 방법이 없기 때문에 특정 숫자(0x41)로 처리하여 사용
  - ▷  $A = 65 = 0x41 = 0100\ 0001$
  - ▷ 책상 = desk = つくえ

10	16	문자	10	16	문자	10	16	문자	10	16	문자
0	0	Null	33	21	!	66	42	B	99	63	c
1	1	Start Of Heading	34	22	"	67	43	C	100	64	d
2	2	Start Of Text	35	23	#	68	44	D	101	65	e
3	3	End Of Text	36	24	\$	69	45	E	102	66	f
4	4	End Of Line	37	25	%	70	46	F	103	67	g
5	5	Escape	38	26	&	71	47	G	104	68	h
6	6	Acknowledge	39	27	'	72	48	H	105	69	i
7	7	Audible Bell (Beep)	40	28	(	73	49	I	106	70	j
8	8	Back-Space	41	29	)	74	50	J	107	71	k
9	9	Horizontal Tab	42	30	*	75	51	K	108	72	l
10	10	Line Feed	43	31	+	76	52	L	109	73	m
11	11	Vertical Tab	44	32	,	77	53	M	110	74	n
12	12	Form Feed	45	33	-	78	54	N	111	75	o
13	13	Carriage Return	46	34	.	79	55	O	112	76	p
14	14	Shift Out	47	35	/	80	56	P	113	77	q
15	15	Shift In	48	36	:	81	57	Q	114	78	r
16	16	Data Link Escape	49	37	;	82	58	R	115	79	s
17	17	Device Control 1	50	38	<	83	59	S	116	80	t
18	18	Device Control 2	51	39	=	84	60	T	117	81	u
19	19	Device Control 3	52	40	@	85	61	U	118	82	v
20	20	Device Control 4	53	41	A	86	62	V	119	83	w
21	21	Neg. Ack	54	42	B	87	63	W	120	84	x
22	22	Synchronous Idle	55	43	C	88	64	X	121	85	y
23	23	End Trans. Block	56	44	D	89	65	Y	122	86	z
24	24	Cancel	57	45	E	90	66	Z	123	87	{
25	25	End of Medium	58	46	F	91	67	[	124	88	
26	26	Substitution	59	47	G	92	68	\			
27	27	Unit Separator	60	48	H	93	69	]			
28	28	Space Bar	61	49	I	94	70	^			
29	29		62	50	J	95	71	_			
30	30		63	51	K	96	72	`			
31	31		64	52	L	97	73	a			
32	32		65	53	M	98	74	b			

초창기 128개이던 아스키(ASCII) 값

# 자료형의 개요

## ▶ 2진수와 16진수의 활용

### ▶ 16진수로 4bit의 표현이 가능

▷ 0 ~ 15 → 0000 ~ 1111

### ▶ 0 ~ 9숫자와 a ~ f 까지의 문자 사용

10진 수	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16진 수	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

### ▶ 16진수로 2진수를 표기하는 것이 효율적임

▷ 아스키 코드에서 A는 10진수로 65

▷  $A = 65 = 0100\ 0001 = 0x41$



# 자료형의 개요

## ▶ 정수형 자료형

### ▶ int형

- ▷ 정수형 연산의 기본 단위
- ▷ 32비트 컴퓨터에서 가장 빠르게 연산

### ▶ short형

- ▷ int형에 비하여 메모리 적게 사용하는 장점
- ▷ 연산 과정에서 기본 단위인 int형으로 변환
  - ▷ 실행 속도 느림

### ▶ long형

- ▷ 컴파일러에 따라 크기가 다르게 정의
- ▷ int형, long형의 크기가 같은 컴파일러라면 사용 불필요



# 자료형의 개요

## ▶ 자료형의 크기

▶ 개발환경에 따라 자료형의 크기가 다르므로 sizeof 연산자로 크기를 확인

```
#include<stdio.h>
int main(void) {
    char num1 = 10;
    short num2 = 20;
    int num3 = 30;
    long num4 = 40;

    printf("\n-----정수형 자료형과 변수의 메모리 크기 -----\n");
    printf("char형의 크기 : %d바이트 , 변수 num1의 크기 : %d바이트 \n", sizeof(char), sizeof(num1));
    printf("short형의 크기 : %d바이트 , 변수 num2의 크기 : %d바이트 \n", sizeof(short), sizeof(num2));
    printf("int형의 크기 : %d바이트 , 변수 num3의 크기 : %d바이트 \n", sizeof(int), sizeof(num3));
    printf("long형의 크기 : %d바이트 , 변수 num4의 크기 : %d바이트 \n", sizeof(long), sizeof(num4));
}
```

정수형			
char	short	int	long
1바이트	2바이트	4바이트	4바이트

```
-----정수형 자료형과 변수의 메모리 크기 -----
char형의 크기 : 1바이트 , 변수 num1의 크기 : 1바이트
short형의 크기 : 2바이트 , 변수 num2의 크기 : 2바이트
int형의 크기 : 4바이트 , 변수 num3의 크기 : 4바이트
long형의 크기 : 4바이트 , 변수 num4의 크기 : 4바이트
```

# 자료형의 개요

## ▶ 자료형의 크기

- ▶ 개발환경에 따라 자료형의 크기가 다르므로 sizeof 연산자로 크기를 확인

```
#include<stdio.h>
int main(void) {
    float num5 = 3.14;
    double num6 = 3.15;
    long double num7 = 3.17;

    printf("\n-----실수형 자료 형과 변수의 메모리 크기 -----\n");
    printf("float형의 크기 : %d바이트 , 변수5의 크기 : %d바이트 \n", sizeof(float), sizeof(num5));
    printf("double형의 크기 : %d바이트 , 변수6의 크기 : %d바이트 \n", sizeof(double), sizeof(num6));
    printf("long double형의 크기 : %d바이트 , 변수7의 크기 : %d바이트 \n", sizeof(long double), sizeof(num7));
}
```

실수형		
float	double	long double
4바이트	8바이트	8바이트

```
-----실수형 자료 형과 변수의 메모리 크기 -----
float형의 크기 : 4바이트 , 변수5의 크기 : 4바이트
double형의 크기 : 8바이트 , 변수6의 크기 : 8바이트
long double형의 크기 : 8바이트 , 변수7의 크기 : 8바이트
```

# 자료형의 개요

## ▶ 정수형 종류와 데이터 표현 범위

▶ char(1바이트), short(2바이트), int(4바이트), long(4바이트)

정수형	메모리크기	데이터 표현 범위
char	1바이트 (8비트)	-128 ~ +127
short	2바이트 (16비트)	-32768 ~ +32767
int	4바이트 (32비트)	-2147483648 ~ +2147483647
long	4바이트 (32비트)	-2147483648 ~ +2147483647

## ▶ 데이터의 표현 범위를 구하는 공식

n은 비트 수(1바이트는 8비트)

$$-2^{n-1}$$

최솟값(MIN)

~

$$+2^{n-1}-1$$

최댓값(MAX)

# 자료형의 개요

## ▶ 자료형의 처리 범위보다 큰 값을 저장하는 경우

### ▶ 비정상적인 처리 결과를 출력

#### ▷ 오버플로우(overflow)와 언더플로우(underflow)는 순환된 값을 출력

```
#include <stdio.h>
int main(void) {
    char overNum = 128;           // 최댓값(127)보다 +1만큼 큰 값 저장(오버플로우)
    char underNum = -129;         // 최솟값(-128)보다 -1만큼 작은 값 저장(언더플로우)

    printf("%d \n", overNum);     // -128 출력
    printf("%d \n", underNum);    // 127 출력

    overNum = 129;                // 최댓값(127)보다 +2만큼 큰 값 저장(오버플로우)
    underNum = -130;              // 최솟값(-128)보다 -2만큼 작은 값 저장(언더플로우)

    printf("%d \n", overNum);     // -127 출력
    printf("%d \n", underNum);    // 126 출력

    return 0;
}
```

-128  
127  
-127  
126

# 자료형의 개요

## ▶ 자료형의 표현 범위에 따른 순환값 출력

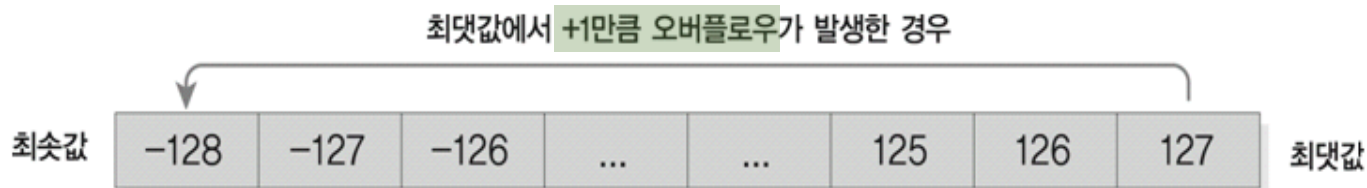
### ▶ Overflow

▶ 자료형에 저장할 수 있는 최대 범위보다 큰 수를 대입하는 경우 발생

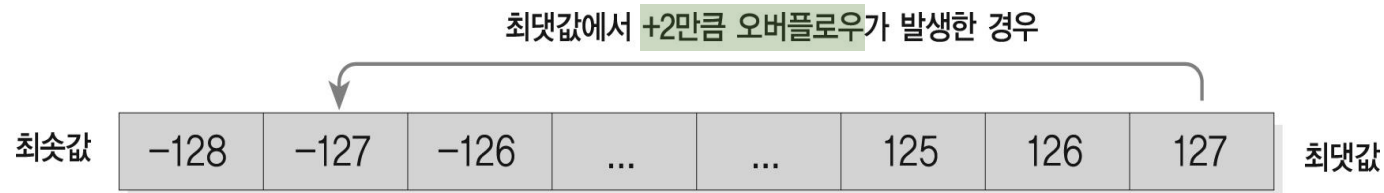
### ▶ 앞의 예제에서는 오버플로우 발생

▶ char 자료형은 1byte이고 데이터 표현 범위는 -128 ~ 127

▶ char overNum = 128;                      // 최댓값보다 +1만큼 큰 값 저장(오버플로우)



▶ overNum = 129;                      // 최댓값(127)보다 +2만큼 큰 값 저장(오버플로우)



# 자료형의 개요

## ▶ 자료형의 표현 범위에 따른 순환값 출력

### ▶ Underflow

▶ 자료형에 저장할 수 있는 최소 범위보다 작은 수를 저장하는 경우 발생

### ▶ 앞의 예제에서는 오버플로우 발생

▶ char 자료형은 1byte이고 데이터 표현 범위는 -128 ~ 127

▶ char underNum = -129;                    // 최솟값보다 -1만큼 작은 값 저장(언더플로우)



▶ underNum = -130;                    // 최솟값(-128)보다 -2만큼 작은 값 저장(언더플로우)



# 자료형의 개요

## ▶ limits.h

▶ 정수형 데이터 표현 범위를 자동으로 알려주는 라이브러리

▶ 데이터 표현의 최솟값(MIN)과 최댓값(MAX) 상수 제공

정수형	메모리크기	데이터 표현 범위	상수(최솟값)	상수(최댓값)
char	1바이트 (8비트)	-128 ~ +127	CHAR_MIN	CHAR_MAX
short	2바이트 (16비트)	-32768 ~ +32767	SHRT_MIN	SHRT_MAX
int	4바이트 (32비트)	-2147483648 ~ +2147483647	INT_MIN	INT_MAX
long	4바이트 (32비트)	-2147483648 ~ +2147483647	LONG_MIN	LONG_MAX

```
#include <stdio.h>
#include <limits.h> // 정수형의 최솟값(MIN), 최댓값(MAX) 상수 정의
int main(void){
    printf("char의 최솟값 %d, 최댓값 %d \n", CHAR_MIN, CHAR_MAX);
    printf("short의 최솟값 %d, 최댓값 %d \n", SHRT_MIN, SHRT_MAX);
    printf("int의 최솟값 %d, 최댓값 %d \n", INT_MIN, INT_MAX);
    printf("long의 최솟값 %d, 최댓값 %d \n", LONG_MIN, LONG_MAX);
    return 0;
}
```

char의 최솟값 -128, 최댓값 127  
short의 최솟값 -32768, 최댓값 32767  
int의 최솟값 -2147483648, 최댓값 2147483647  
long의 최솟값 -2147483648, 최댓값 2147483647

# 자료형의 개요

## ▶ unsigned 자료형

▶ unsigned: 0과 양수만을 표현

▶ 정수형 저장 시 양수만 저장하면 범위를 두 배 더 넓게 저장 가능

▶  $-2^{\text{비트수}-1} \sim 2^{\text{비트수}-1} - 1 \rightarrow 0 \sim 2^{\text{비트수}} - 1$

▶ char 자료형의 예

▶  $-2^7 \sim 2^7 - 1 \rightarrow 0 \sim 2^8 - 1$

▶  $-128 \sim 127 \rightarrow 0 \sim 255$

정수형	메모리크기	데이터 표현 범위
char (signed char) unsigned char	1바이트 (8비트) 1바이트 (8비트)	-128 ~ 127 0 ~ (128 + 127)
short (signed short) unsigned short	2바이트 (16비트) 2바이트 (16비트)	-32768 ~ 32767 0 ~ (32768 + 32767)
int (signed int) unsigned int	4바이트 (32비트) 4바이트 (32비트)	-2147483648 ~ 2147483647 0 ~ (2147483648 + 2147483647)
long (signed long) unsigned long	4바이트 (32비트) 4바이트 (32비트)	-2147483648 ~ 2147483647 0 ~ (2147483648 + 2147483647)



# 자료형의 개요

## ▶ unsigned 자료형

- ▶ unsigned 없으면 자동으로 signed 선언
- ▶ unsigned 변수 사용시 주의점
  - ▷ 큰 양수 저장 후 %d로 출력하면 음수 출력
  - ▷ 음수 저장 후 %u로 출력하면 양수 출력

```
#include <stdio.h>
int main(void){
    unsigned int a;

    a = 4294967295;           //232-1
    printf("%d \n", a);
    a = -1;
    printf("%u \n", a);
}
```



```
-1
4294967295
```

정수형	메모리 크기	데이터 표현 범위
int (signed int)	4바이트(32비트)	-2147483648 ~ 2147483647
unsigned int	4바이트(32비트)	0 ~ (2147483648 + 2147483647)

# 자료형의 개요

## ▶ unsigned 자료형

▶ 두 정수가 메모리에 저장되는 형태 같으나 printf()함수에서 서식문자로 해석해서 보여주는 방법이 다름

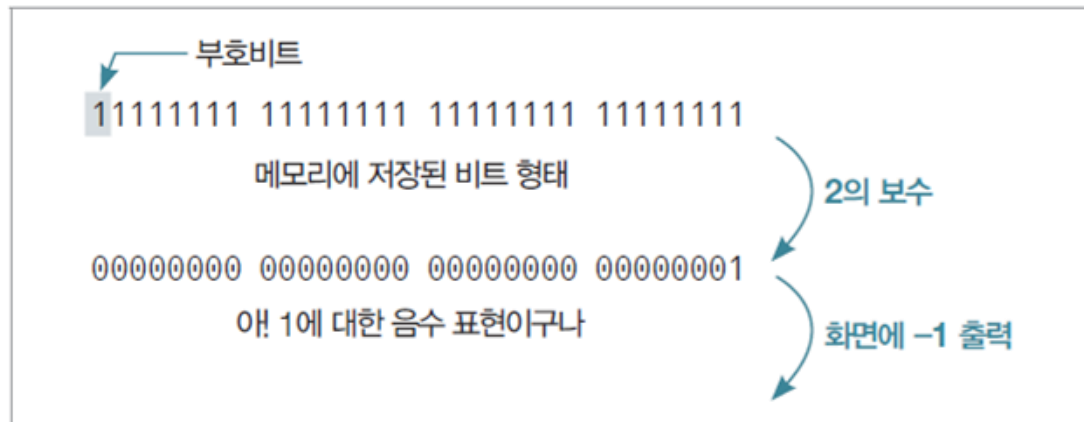
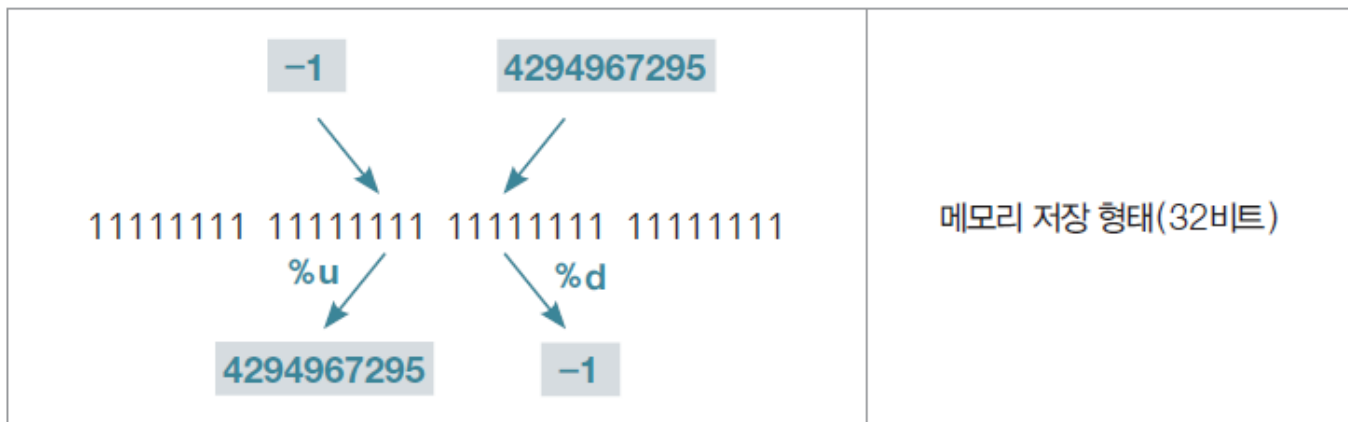
▶ unsigned 자료형 - 항상 양수만 저장하고 %u로 출력

▶ %d로 출력하면 가장 왼쪽 비트를 부호 비트로 간주하지만 %u를 사용하면 단순히 전체를 10진수로 인식

▶ 음수는 입력된 데이터의 절대값을 2의 보수로 저장

▶ 보수는 원래 데이터 값과 반대 비트(bit)의 값

▶ 2의 보수는 원래 데이터 값과 더하였을 경우, 결과가 0이 되는 값



# 자료형의 개요

## ▶ unsigned 자료형

▶ limits.h 에서 제공하는 unsigned형 상수의 최댓값(MAX)

unsigned 정수형	데이터 표현 범위	상수(최댓값)
unsigned char	0 ~ (127 + 128)	UCHAR_MAX
unsigned short	0 ~ (32767 + 32768)	USHRT_MAX
unsigned int	0 ~ (2147483647 + 2147483648)	UINT_MAX
unsigned long	0 ~ (2147483647 + 2147483648)	ULONG_MAX

```
#include <stdio.h>
#include <limits.h>
int main(void) {
    printf("unsigned char 최댓값 : %u \n", UCHAR_MAX);
    printf("unsigned short 최댓값 : %u \n", USHRT_MAX);
    printf("unsigned int 최댓값 : %u \n", UINT_MAX);
    printf("unsigned long 최댓값 : %u \n", ULONG_MAX);
    return 0;
}
```

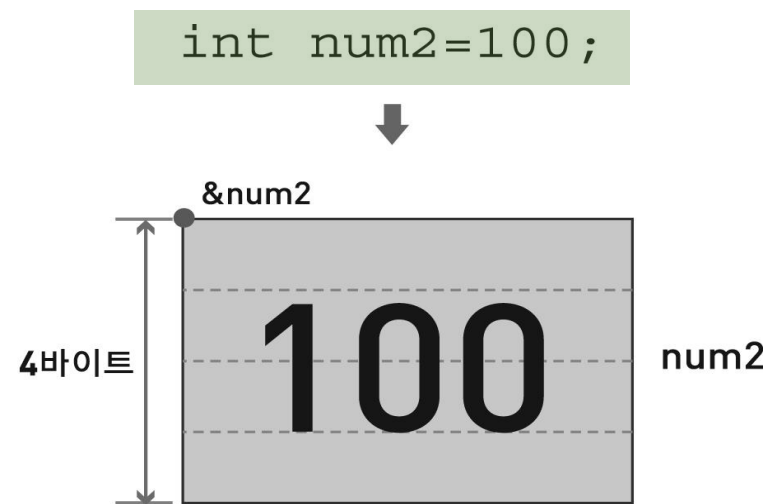
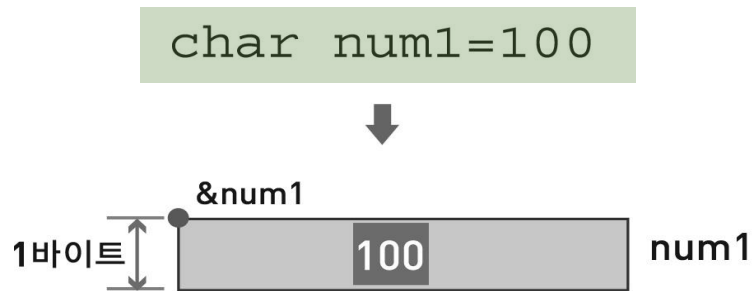
Microsoft Visual Studio 디버그 콘솔

```
unsigned char 최댓값 : 255
unsigned short 최댓값 : 65535
unsigned int 최댓값 : 4294967295
unsigned long 최댓값 : 4294967295
```

# 자료형의 개요

▶ 정수형은 int형을 선호

▶ char형 변수와 int형 변수의 차이



▶ CPU가 int형을 가장 빠르게 처리하는 이유

▶ 개발된 대부분의 컴퓨터들은 32비트 이상의 시스템

▶ CPU가 연산하는 기본 단위가 최소 32비트

# 자료형의 개요

## ▶ 변수의 시작 주소와 &연산자

### ▶ 변수의 시작주소

- ▶ 변수 앞에 &연산자를 붙이면 변수의 시작 주소를 알 수 있음
- ▶ printf함수에서 %x 서식문자를 사용하면 데이터를 16진수로 출력
  - ▶ 메모리 주소를 출력할 경우, 주로 16진수로 표기

```
#include <stdio.h>
int main(void){
    int a = 3;
    int b = 4;
    printf("a의 값: %d \n", a);
    printf("b의 값 : %d \n", b);

    printf("변수 a의 짝 주소: %x \n", &a);
    printf("변수 b의 짝 주소: %x \n", &b);
    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

```
a의 값: 3
b의 값 : 4
변수 a의 시작 주소: 1ff9c8
변수 b의 시작 주소: 1ff9bc
```

# 자료형의 개요

## ▶ 실수형

▶ 실수형 데이터를 저장하는 변수의 자료형

▶ 소수점을 가진 실수의 값을 표현할 수 있는 자료형

▷ 항공 제어 시스템과 같이 정확한 데이터가 필요한 환경에서는 정밀한 계산을 위하여 사용

▶ 정수형과 마찬가지로 크기에 따라 값을 저장할 수 있는 범위가 달라짐

실수형	메모리크기	데이터 표현 범위	유효숫자	변환문자(서식문자)
float	4바이트 (32비트)	$-3.4 * 10^{38} \sim 3.40 * 10^{38}$	7	%f
double	8바이트 (64비트)	$-1.79 * 10^{308} \sim 1.79 * 10^{308}$	15	%lf

▶ 유효숫자가 많을수록 더 정확한 값 표현 가능

▶ 변수는 정수형을 기본적으로 사용

▷ 실수형은 꼭 필요한 경우만 사용


# 자료형의 개요

## ▶ 실수형

- ▶ 컴파일러는 기본적으로 실수형을 double로 인식
  - ▷ 오차를 줄이고 정밀도를 높이기 위해서
- ▶ 값이 훼손 될 수 있으므로 가능하면 실수를 다루는 경우에는 유효숫자가 많은 double 형을 사용
- ▶ 부득이하게 float 타입을 사용할 경우, 저장 할 때 '1.23f' 나 '1.23F'처럼 에프를 붙여서 사용
  - ▷ 이는 실수 리터럴이 float 타입임을 나타냄

```
#include<stdio.h>
int main(void){
    float num1 = 0.123456;           // float num1=0.123456F;
    printf("float형 : %f \n", num1);  // 0.123456(소수점 6자리까지 출력)
    printf("float형 : %.2f \n", num1); // 0.12(소수점 2자리까지 출력)
    return 0;
}
```

warning C4305: '초기화 중': 'double'에서 'float'(으)로 잘립니다.

 Microsoft Visual Studio 디버그 콘솔

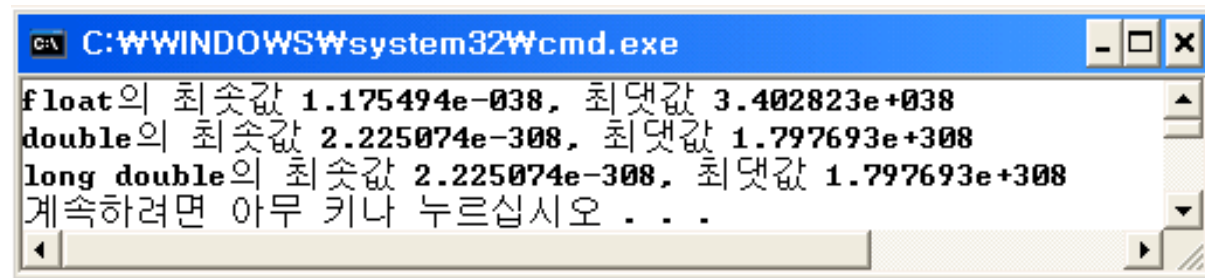
```
float형 : 0.123456
float형 : 0.12
```

# 자료형의 개요

## ▶ float.h

- ▶ 실수형 데이터 표현 범위를 자동으로 알려주는 라이브러리
- ▶ 데이터 표현 최솟값(MIN)과 최댓값(MAX) 상수 제공

```
#include <stdio.h>
#include <float.h> // 실수형의 데 터 표현 범 위 상수 정의
int main(void){
    printf(" float의 최솟값 %e, 최댓값 %e \n", FLT_MIN, FLT_MAX);
    printf("double의 최솟값 %e, 최댓값 %e \n", DBL_MIN, DBL_MAX);
    printf("long double의 최솟값 %e, 최댓값 %e \n", LDBL_MIN, LDBL_MAX);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
float의 최솟값 1.175494e-038, 최댓값 3.402823e+038
double의 최솟값 2.225074e-308, 최댓값 1.797693e+308
long double의 최솟값 2.225074e-308, 최댓값 1.797693e+308
계속하려면 아무 키나 누르십시오 . . .
```



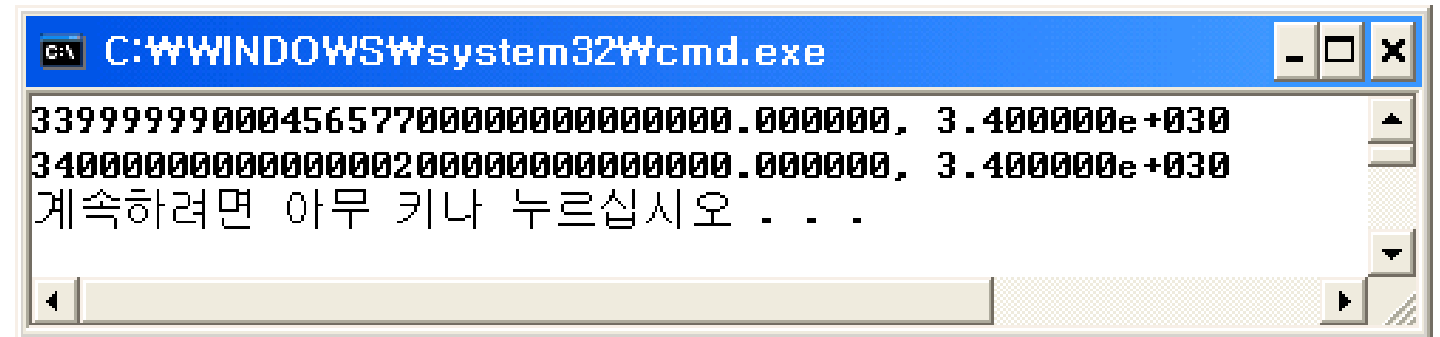
# 자료형의 개요

## ▶ 실수형을 지수형으로 표현

### ▶ %e와 %le 서식문자를 사용

```
#include <stdio.h>
int main(void){
    float num1=3.4e+30;
    double num2=3.4e+30;

    printf("%f, %e \n", num1, num1);    // float형 오차발생
    printf("%lf, %le \n", num2, num2);  // double형은 정상
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
3399999900045657700000000000000.000000, 3.400000e+030
3400000000000000020000000000000.000000, 3.400000e+030
계속하려면 아무 키나 누르십시오 . . .
```

# 자료형의 개요

## ▶ 실수형의 소수점 자릿수 표현

### ▶ %과 f(lf)사이에 표현할 자릿수를 .과 함께 입력

```
#include<stdio.h>
int main(void){
    float num1=0.123456789012345;
    double num2=0.123456789012345;

    printf("float형 : %f \n", num1);          // 0 .123457 출력
    printf("double형 : %lf \n", num2);        // 0 .123457 출력

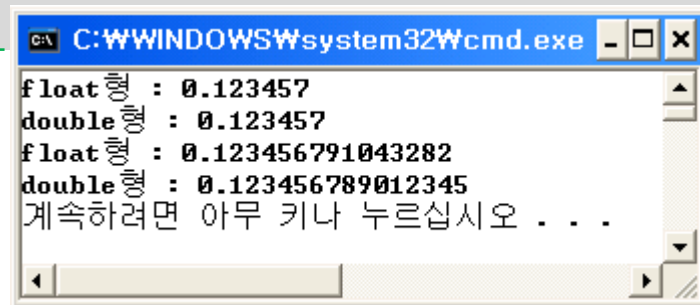
    printf("float형 : %.15f \n", num1);       // 0.123456791043282 출력
    printf("double형 : %.15lf \n", num2);     // 0.123456789012345 출력

    return 0;
}
```

### ▶ 유효숫자

▶ float : 소수점 7째 자리

▶ double : 소수점 15째 자리



```
C:\WINDOWS\system32\cmd.exe
float형 : 0.123457
double형 : 0.123457
float형 : 0.123456791043282
double형 : 0.123456789012345
계속하려면 아무 키나 누르십시오 . . .
```

# 형변환

## ▶ 자료형 변환의 종류

- ▶ 자동 형변환 : 서로 다른 자료형의 피연산자로 연산을 할 경우에 컴파일러가 자동으로 타입을 변환하는 것

```
int a = 3.14;  
double b = 10;
```

- ▶ 강제 형변환 : 프로그래머가 형변환에 관련된 코드를 명시하여 강제로 타입을 변환하는 것

```
int a = 10;  
int b = 20;  
double c = (double)a * (double)b;
```

# 형변환

## ▶ 자동 형변환

### ▶ 다른 자료형 간 산술 연산의 경우에 작은형에서 큰형으로 자동 형변환

▷ 정수 + 실수 또는 실수 + 정수와 같은 산술 연산을 하는 경우

### ▶ 자료형 변환 우선순위 (작은형에서 큰형으로...)

▷  $\text{char} < \text{int} < \text{long} < \text{float} < \text{double} < \text{long double}$

▷ 실수 리터럴을 정수형으로 변경하는 경우 소수점 이하는 버림(손실)

### ▶ 대입 연산을 하는 경우

▷ 대입 연산자를 기준으로 오른쪽에서 왼쪽으로 자동 형변환

```
#include <stdio.h>
int main(void){
    int    num1 = 100;           // 정수
    double num2 = 3.14;         // 실수

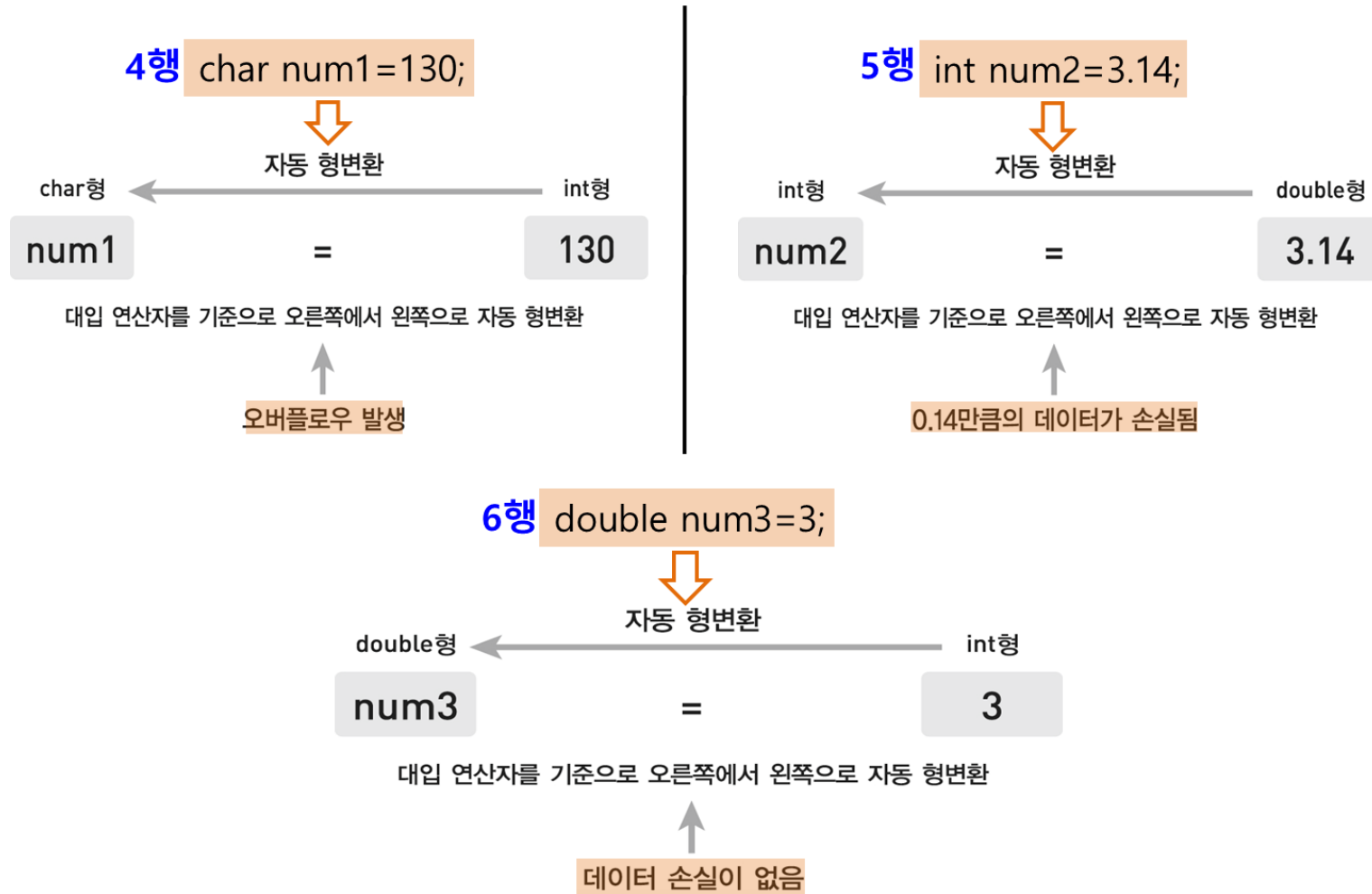
    printf("%lf \n", num1 + num2); // 정수 + 실수
    return 0;
}
```

```
#include <stdio.h>
int main(void){
    char    num1 = 130;
    int     num2 = 3.14;
    double  num3 = 3;

    printf("%d, %d, %lf \n", num1, num2, num3);
    return 0;
}
```

# 형변환

## ▶ 자동 형변환



# 형변환

## ▶ 자동 형변환

### ▶ 데이터 손실이 없는 경우

▷ 예) int형 (작은 자료형) 에서 double형(큰 자료형)으로 변환되는 경우

### ▶ 데이터 손실이 있는 경우

▷ 예) double형 (큰 자료형) 에서 int형(작은 자료형)으로 변환되는 경우

▷ 연산 결과에 따라 데이터 손실이 있는 경우에는 강제형변환을 통하여 문제점을 해결

# 자료형의 개요

## ▶ 강제 형변환

▶ 이미 정의된 자료형을 강제로 다른 자료형으로 변환하는 것

▶ 괄호 연산자 ( )를 이용

```
int num1=2;  
(double) num1;
```

↑            ↑  
자료형       변수

```
#include <stdio.h>  
int main(void){  
    int num1=10, num2=3;  
    double result;  
  
    result=num1/num2;  
    printf("결과 : %lf \n", result);    //결과 : 3.000000  
  
    result=(double)num1/num2;  
    printf("결과 : %lf \n", result);    //결과 : 3.333333  
  
    result=num1/(double)num2;  
    printf("결과 : %lf \n", result);    //결과 : 3.333333  
  
    result=(double)num1/(double)num2;  
    printf("결과 : %lf \n", result);    //결과 : 3.333333  
    return 0;  
}
```

---

# Q & A



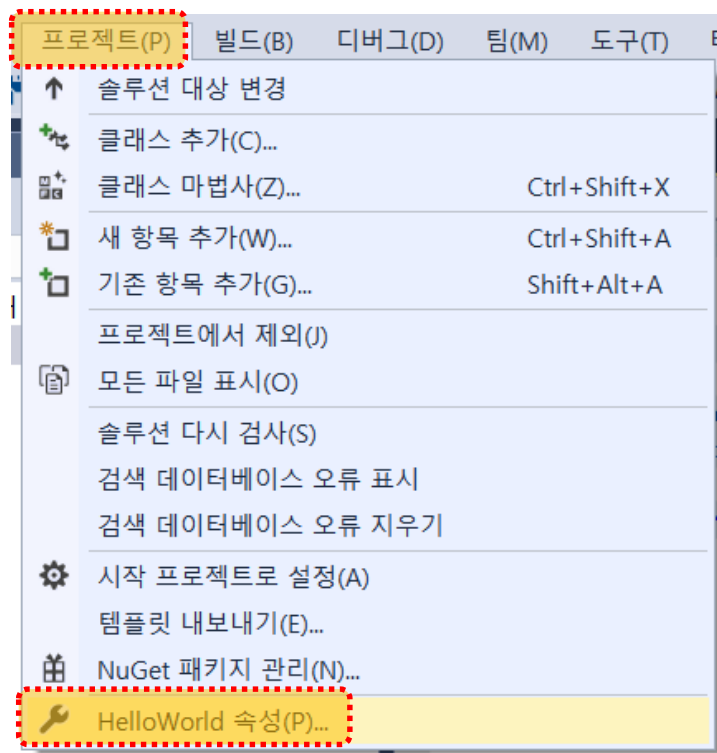
# 비주얼 스튜디오 설정

## ▶ SDL 설정

▶ 원활한 예제 진행을 위하여 SDL(Security Development Lifecycle)해제

▶ scanf()함수 사용 가능

▶ [프로젝트] - [속성] - [C/C++] - [일반] - [SDL 검사] - “아니오”로 설정



HelloWorld 속성 페이지

