C 프로그래밍 및 실습

C Programming

이지민

CHAP 03 데이터형과 변수

학습목표

변수와 상수의 개념에 대해 알아본다.

리터럴 상수, 매크로 상수, const변수에 대해 알아본다.

C 언어의 자료형에 대해 알아본다.

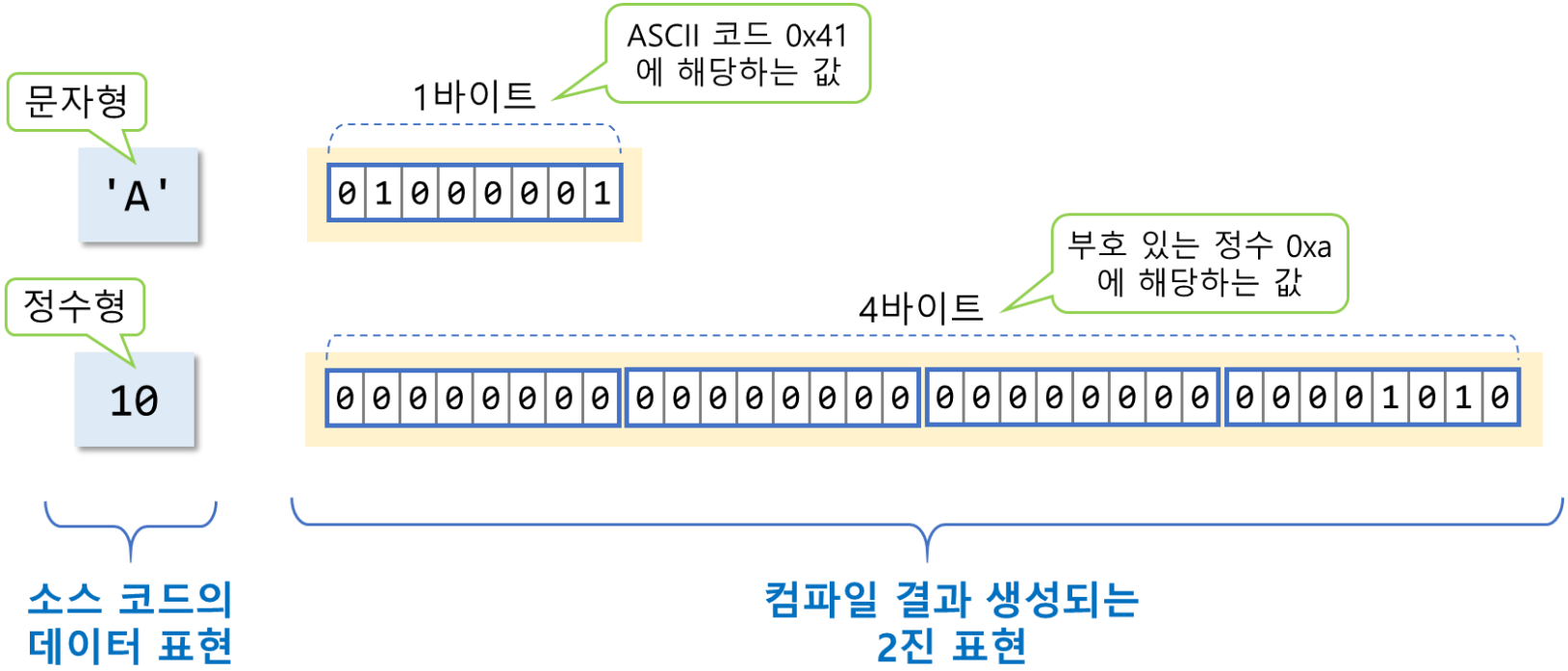
목차

- 자료형
  - 자료형(데이터형)의 개념
  - 정수형
  - 문자형
  - 실수형
- 변수와 상수
  - 변수
  - 상수

자료형_컴퓨터 데이터표현 방식

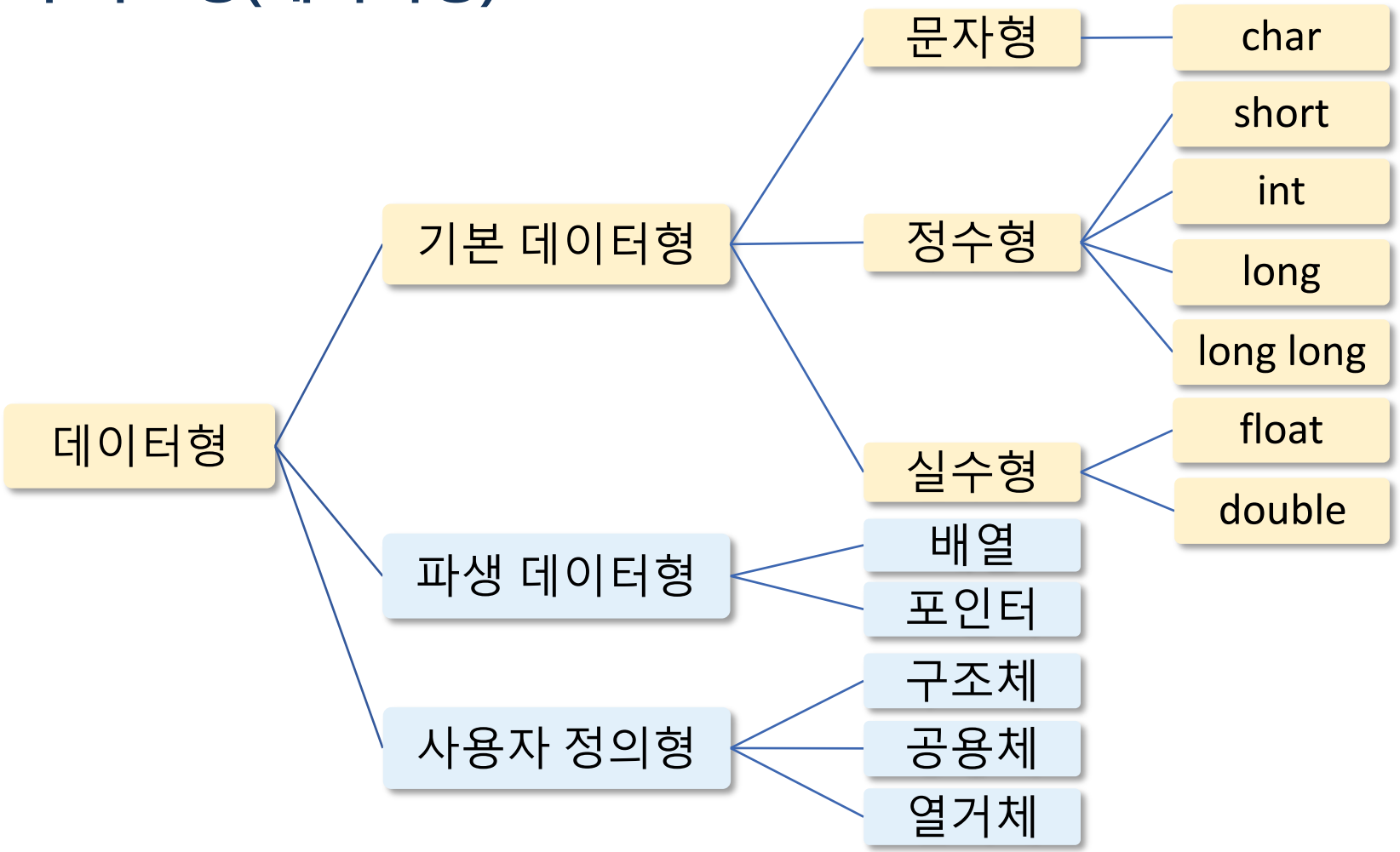
❖데이터의 2진 표현

- **컴퓨터 시스템에서 2진 데이터로 값을 표현하고 저장하는 방식**
  - 모든 값에는 데이터형이 있다.
- **컴파일러는 데이터형에 따라, 값을 저장하는데 필요한 메모리의 크기와 2진 표현을 결정한다.**



자료형_C의 데이터 형

❖C의 자료형(데이터형)



자료형_데이터형의 바이트 크기

❖sizeof 연산자

- 데이터형이나 어떤 값의 바이트 크기를 구하려면 **sizeof 연산자**를 이용한다.
- 데이터형의 크기는 플랫폼에 따라 다르다.

형식	sizeof(데이터형) sizeof(값)
사용예	sizeof(short) sizeof(123)

- sizeof 연산자는 데이터형뿐만 아니라 변수나 값에도 사용할 수 있다.

```
int a;
printf("size of a : %d\n", sizeof(a));
printf("size of 3.14 : %d\n", sizeof(3.14));
```

변수 a의 바이트 크기를 구한다.

실수값 3.14의 바이트 크기를 구한다.

자료형_기본 데이터형의 크기

❖데이터 형의 크기 구하기(1/2)

```
03  int main(void)
04  {
05      char ch;
06      int num;
07      double x;
08
09      printf("char형의 바이트 크기: %d\n", sizeof(char));
10
11      printf("short형의 바이트 크기: %d\n", sizeof(short));
12      printf("int형의 바이트 크기: %d\n", sizeof(int));
13      printf("long형의 바이트 크기: %d\n", sizeof(long));
14      printf("long long형의 바이트 크기: %d\n", sizeof(long long));
15
16      printf("float형의 바이트 크기: %d\n", sizeof(float));
17      printf("double형의 바이트 크기: %d\n", sizeof(double));
18      printf("long double형의 바이트 크기: %d\n", sizeof(long double));
```

여러 가지 데이터형의 변수 선언

데이터형의 바이트 크기 구하기

자료형_기본 데이터형의 크기

❖데이터 형의 크기 구하기(2/2)

```
19
20      printf("ch 변수의 바이트 크기: %d\n", sizeof ch);
21      printf("num 변수의 바이트 크기: %d\n", sizeof num);
22      printf("x 변수의 바이트 크기: %d\n", sizeof x);
23
24      return 0;
25  }
```

변수의 바이트 크기를 구할 수도 있다.

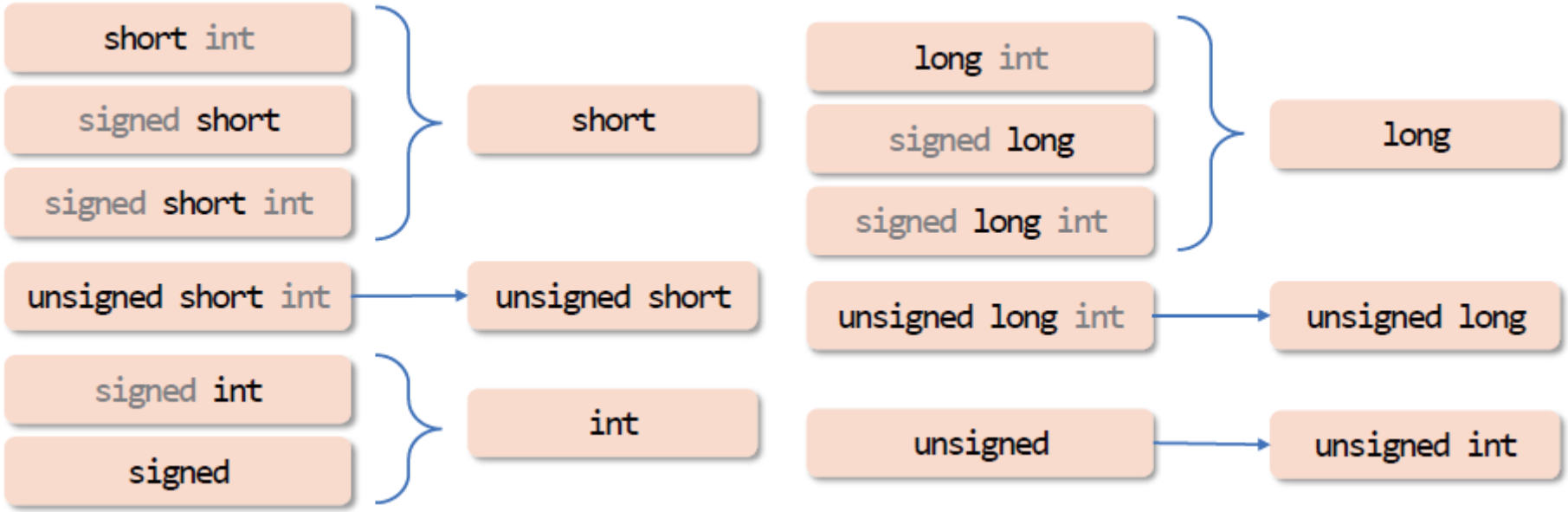
실행결과

```
char형의 바이트 크기: 1
short형의 바이트 크기: 2
int형의 바이트 크기: 4
long형의 바이트 크기: 4
long long형의 바이트 크기: 8
float형의 바이트 크기: 4
double형의 바이트 크기: 8
long double형의 바이트 크기: 8
ch 변수의 바이트 크기: 1
num 변수의 바이트 크기: 4
x 변수의 바이트 크기: 8
```

기본 데이터형의 바이트 크기

자료형_정수형 :종류

- 다양한 크기의 정수형을 제공
  - sizeof(short) ≤ sizeof(int) ≤ sizeof(long)
- 부호 있는 정수형과 부호 없는 정수형으로 구분
  - signed는 생략 가능
  - unsigned : 부호 없는 정수형



자료형_정수형 :종류

❖정수형의 종류

부호 있는 정수형

C99에 추가된 데이터형

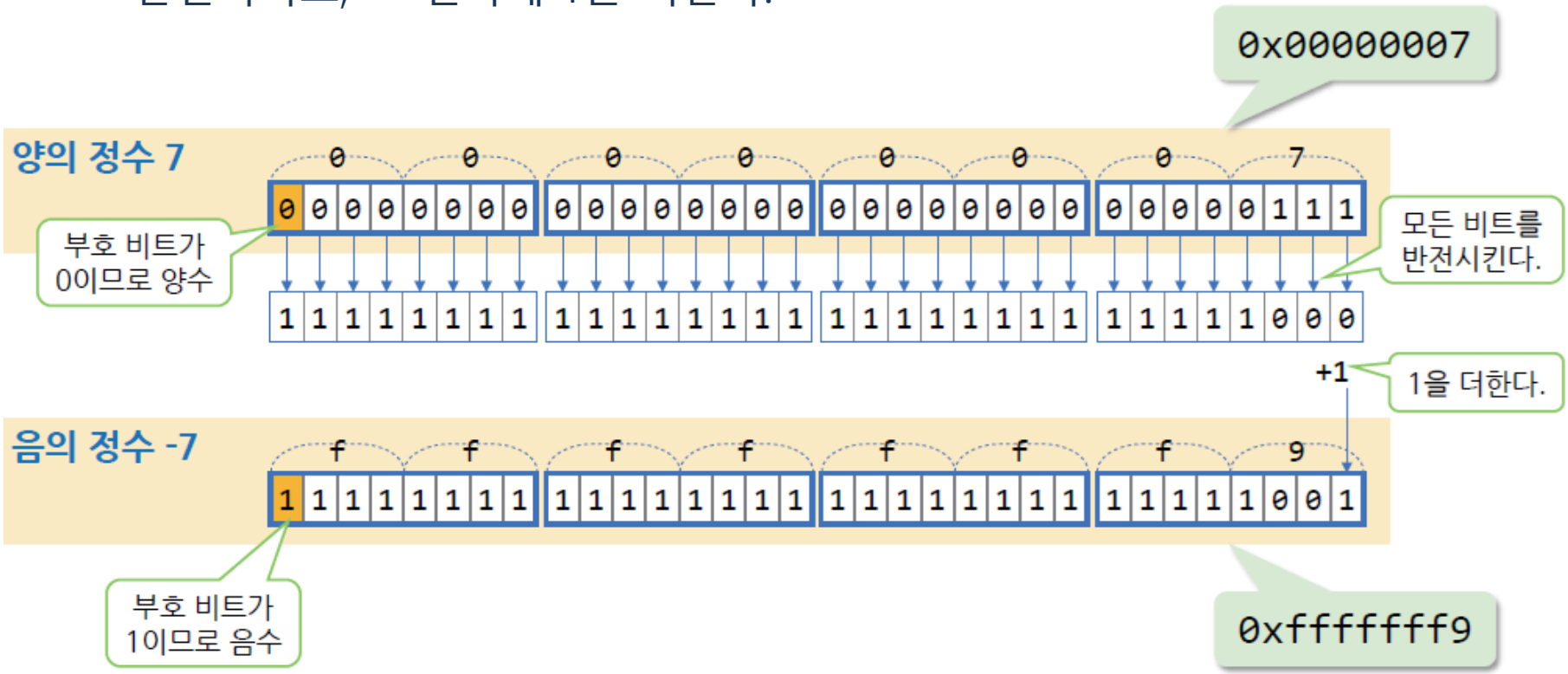
부호 없는 정수형

데이터형	크기	유효 범위
char	1	-128~127
short	2	-32768~32767
int	4	-2147483648~2147483647
long	4	-2147483648~2147483647
long long	8	-9223372036854775808 ~ 9223372036854775807
unsigned char	1	0~255
unsigned short	2	0~65535
unsigned int	4	0~4294967295
unsigned long	4	0~4294967295
unsigned *long long*	8	0~18446744073709551615

자료형_정수형 :2진표현(1/3)

❖부호 있는 정수형 : 2의 보수로 음수 표현

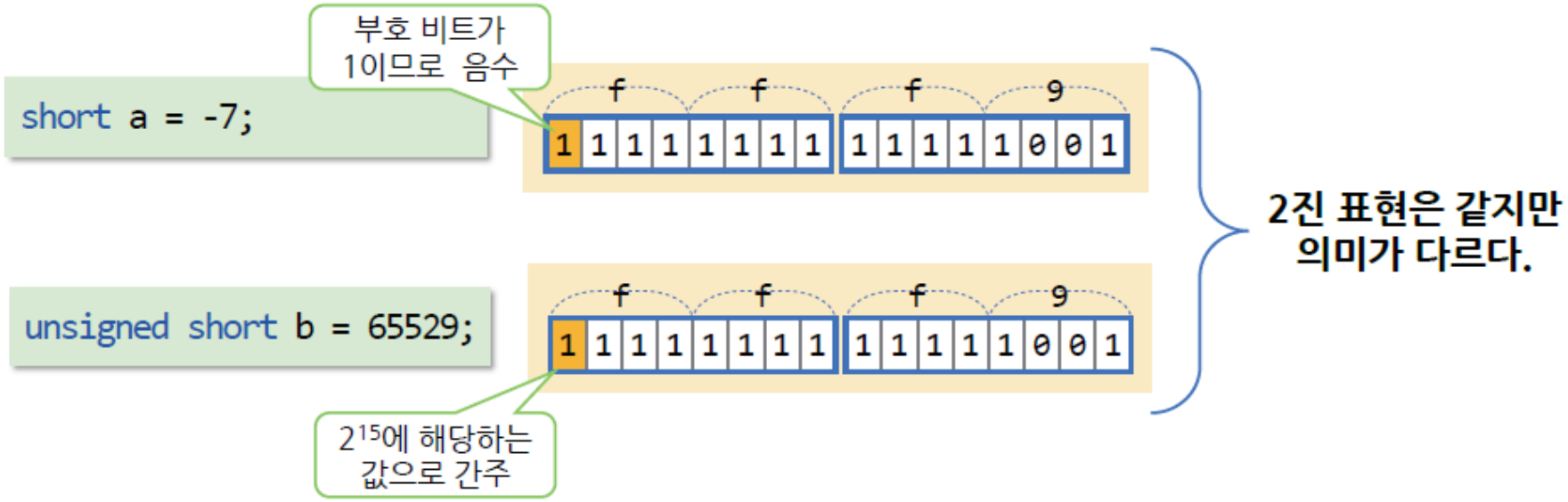
- 최상위 비트는 부호 비트(sign bit)로 사용
- 음수표현은 2의 보수 사용
  - -n을 표현하려면, n을 2진수로 나타낸 다음 각 비트를 0은 1로, 1은 0으로 반전시키고, 그 결과에 1을 더한다.



자료형_정수형 :2진표현(2/3)

❖부호 없는 정수형

- 최상위 비트를 값을 저장하는 용도로 사용



변수의 데이터형이 변수에 저장된 값의 의미를 결정한다.

자료형_정수형 :2진표현(3/3)

❖signed와 unsigned 비교

short형 변수를 출력할 때 4바이트 크기로 변환하므로 하위 2바이트만 비교해야 한다.

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      short x = -7;
06      unsigned short y = 65529;
07
08      printf("x = %5d, %08x\n", x, x);
09      printf("y = %5d, %08x\n", y, y);
10  }
```

실행 결과

```
x =    -7, fffffff9
y = 65529, 0000fff9
```

%08x는 8문자 폭에 맞춰서 16진수로 출력(빈칸에 0 출력)

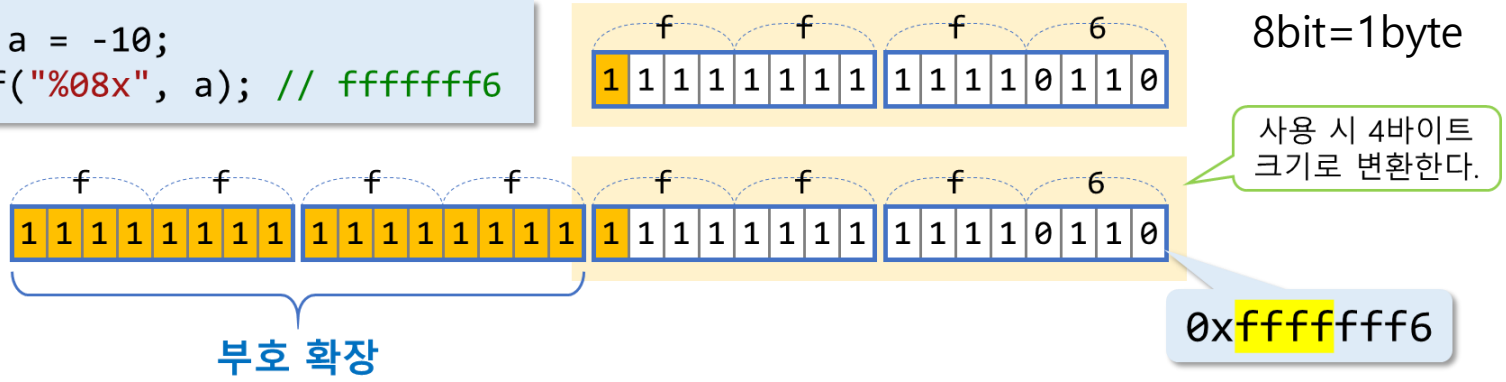
short형 변수는 출력 시 4바이트로 변환

자료형_정수형 :승격

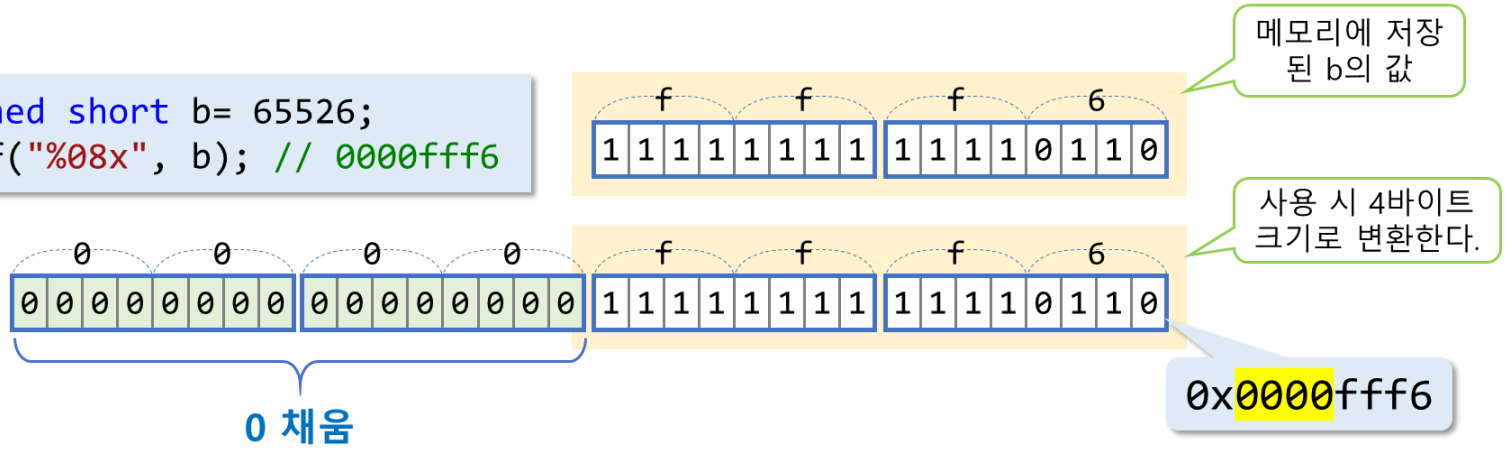
❖정수형의 승격

- char, short형 변수는 사용 시 int형으로 변환된다.

```
short a = -10;
printf("%08x", a); // fffffff6
```




```
unsigned short b= 65526;
printf("%08x", b); // 0000fff6
```



자료형_정수형

❖정수형으로 사용되는 char형

- **char형은 1바이트 크기의 정수형으로도 사용될 수 있다.**
  - 작은 크기의 정수를 저장할 때 유용

정수형으로 사용되는 char형

```
char n = 97;
printf("n   = %d\n", n);
printf("n+1 = %d\n", n + 1);
```

10진수 정수로 출력

정수처럼 덧셈 연산

  - unsigned char형은 1바이트 크기의 2진 데이터를 저장할 때 주로 사용

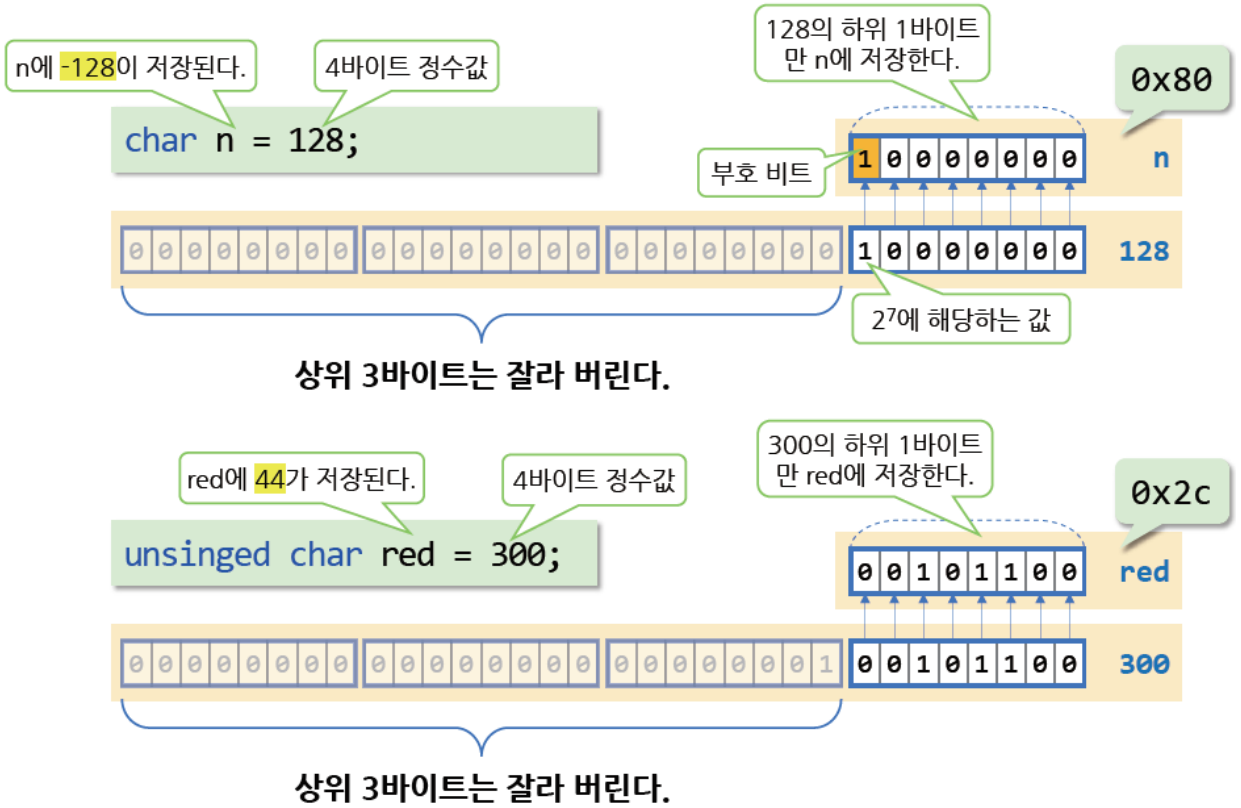
```
unsigned char flags = 0x81;
```

2진 데이터 1000 0001

자료형_정수형 :유효범위

❖정수형의 유효범위

- 정수형의 크기에 따라 표현 가능한 정수의 범위가 달라진다.
- 정수형 변수에 유효 범위 밖의 값을 저장하면, 정수형의 크기에 맞춰 값의 나머지 부분을 잘라 버리고 유효 범위 내의 값만 저장된다.



자료형_정수형 :오버플로우와 언더플로우(1/3)

❖오버플로우(overflow)

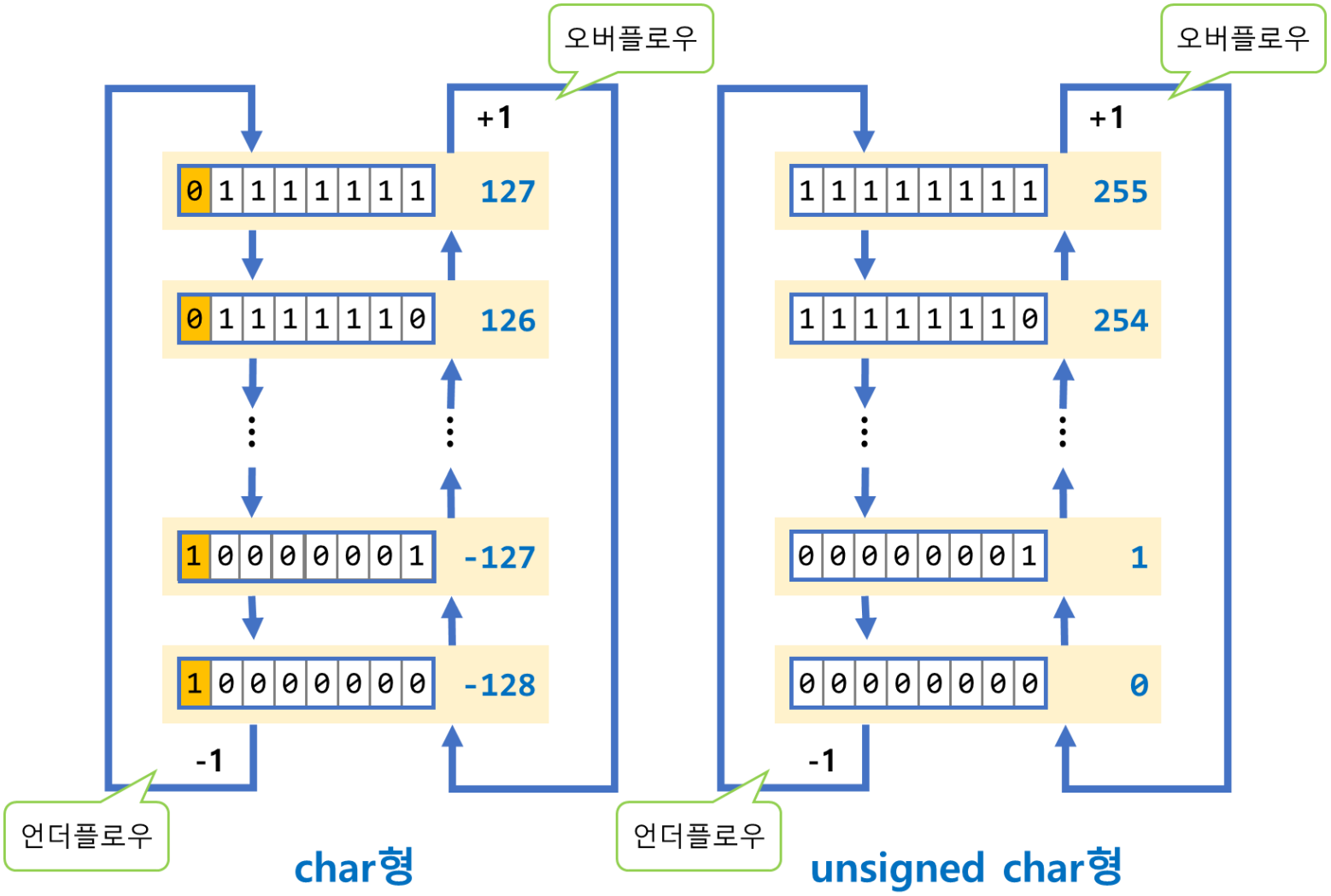
- 정수형 변수에 유효 범위 밖의 값을 저장하면, 유효 범위 내의 값으로 설정되는 것
- 유효 범위를 벗어나는 값을 저장하면 정수형의 크기에 맞춰 나머지 부분을 잘라 버린다.
- 항상 유효 범위 내의 값만 저장된다.

```
unsigned char red = 300;     // 오버플로우가 발생하므로 red에 실제로는 44가 저장된다.
```

❖언더플로우

- 정수형의 최소값보다 작은 값을 저장할 때도 유효 범위 내의 값으로 설정되는 것

자료형_정수형 :오버플로우와 언더플로우(2/3)



자료형_정수형 :오버플로우와 언더플로우(3/3)

❖char형의 오버플로우 예제

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      char n = 128;
06      unsigned char red = 300;
07
08      printf("n = %d\n", n);
09      printf("red = %d\n", red);
10  }
```

실행 결과

```
n = -128
red = 44
```

오버플로우가 발생하여 유효 범위 내의 값으로 설정된다.

char형의 유효 범위를 벗어나는 값을 저장한다.

unsigned char형의 유효 범위를 벗어나는 값을 저장한다.

자료형_문자형 :2진표현

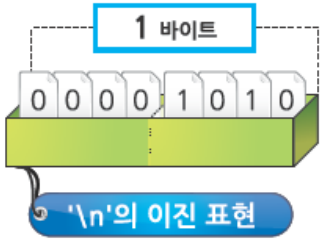
❖문자형

- **각 문자에 대응되는 문자 코드 사용**
  - ASCII 코드, EBCDIC 코드, 한글 완성형 코드 등
  - char형의 변수에 'A' 문자를 저장하면, 실제로는 'A' 문자의 ASCII 코드인 65(0x41)가 저장된다.
- **문자의 2진 표현 : ASCII 코드**
  - 제어 문자 : 0~31, 127
  - 출력 가능한 문자 : 32~126

❖특수 문자

- **ASCII 코드 중 특수 문자는 ' '안에 역슬래시(₩)와 함께 정해진 문자를 이용해서 표현**
- **'₩' 다음에 8진수로 적어주거나 '₩x' 다음에 16진수로 적어줌**

```
char newline1 = '\012';     // newline1에 '\n'을 저장한다.
char newline2 = '\xa';      // newline2에 '\n'을 저장한다.
```



자료형_문자형

❖입력된 문자의 ASCII 코드 확인(1/2)

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int main(void)
05  {
06      char ch, prev_ch, next_ch;
07
08      printf("문자? ");
09      scanf("%c", &ch);
10
11      prev_ch = ch - 1;
12      next_ch = ch + 1;
```

문자형 변수 선언

문자 입력

char형도 정수형이므로 덧셈이나 뺄셈 가능

자료형_문자형

❖입력된 문자의 ASCII 코드 확인(2/2)

```
14      printf("prev_ch = %c, %d, %#02x\n", prev_ch, prev_ch, prev_ch);
15      printf("ch      = %c, %d, %#02x\n", ch, ch, ch);
16      printf("next_ch = %c, %d, %#02x\n", next_ch, next_ch, next_ch);
17  }
```

0x와 함께 16진수로 출력

문자 출력

ASCII 코드 출력 (10진수)

ASCII 코드 출력 (16진수)

실행 결과

```
문자? M
prev_ch = L, 76, 0x4c
ch      = M, 77, 0x4d
next_ch = N, 78, 0x4e
```

자료형_문자형 :이스케이프 시퀀스(1/2)

- 출력할 수 없는 제어 문자나 문자열 안에서 특별하게 표기해야 하는 문자를 나타내는데 사용
- ' ' 안에 역슬래시(₩)와 정해진 문자로 표현
  - '₩n' : 줄바꿈(newline)
  - '\0' : 널 문자(null)
  - '₩t' : 수평 탭(horizontal tab)
  - '\"' : 큰따옴표
  - '\\' : 역슬래시(back slash)

제어 문자

문자열 안에서 특별하게 표기해야 하는 문자

- '₩' 다음에 ASCII 코드값을 적어줄 수도 있다.

```
char separator = '\xa';
```

'₩x' 다음에 ASCII 코드값을 16진수로 적어준다. ('₩n' 의미)

자료형_문자형 :이스케이프 시퀀스(2/2)

❖특수문자

10진수	8진수	16진수	특수 문자	의미
0	000	00	'\0'	널 문자(null)
7	007	07	'\a'	경고음(bell)
8	010	08	'\b'	백스페이스(backspace)
9	011	09	'\t'	수평 탭(horizontal tab)
10	012	0A	'\n'	줄 바꿈(newline)
11	013	0B	'\v'	수직 탭(vertical tab)
12	014	0C	'\f'	폼 피드(form feed)
13	015	0D	'\r'	캐리지 리턴(carriage return)
34	042	22	'\"'	큰따옴표
39	047	27	'\''	작은따옴표
92	134	5C	'\\'	역슬래시(back slash)

자료형_문자형

❖특수 문자

```
03  int main(void)
04  {
05      char bell = '\b'; // 특수 문자
06      printf("%c프로그램을 시작합니다.\n", bell);    // 경고음 발생
07
08      printf("c:\\work\\chap03\\Ex03_06\\Debug\n"); // 역슬래시 출력
09
10      printf("\t탭 문자를 출력합니다.\n"); // 탭 문자 출력
11
12      return 0;
13  }
```

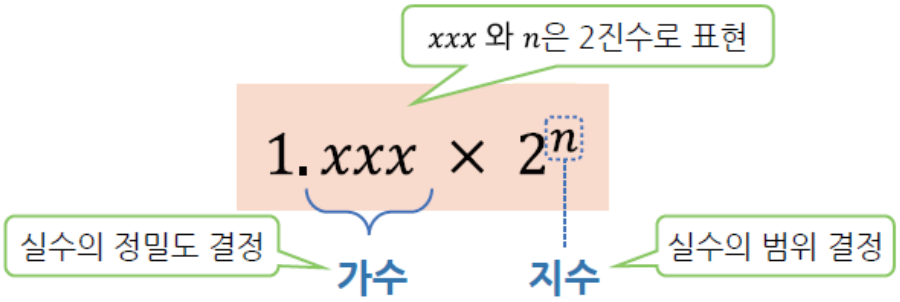
실행결과

```
프로그램을 시작합니다.
c:\work\chap03\Ex03_06\Debug
        탭 문자를 출력합니다.
```

자료형_실수형 :2진표현

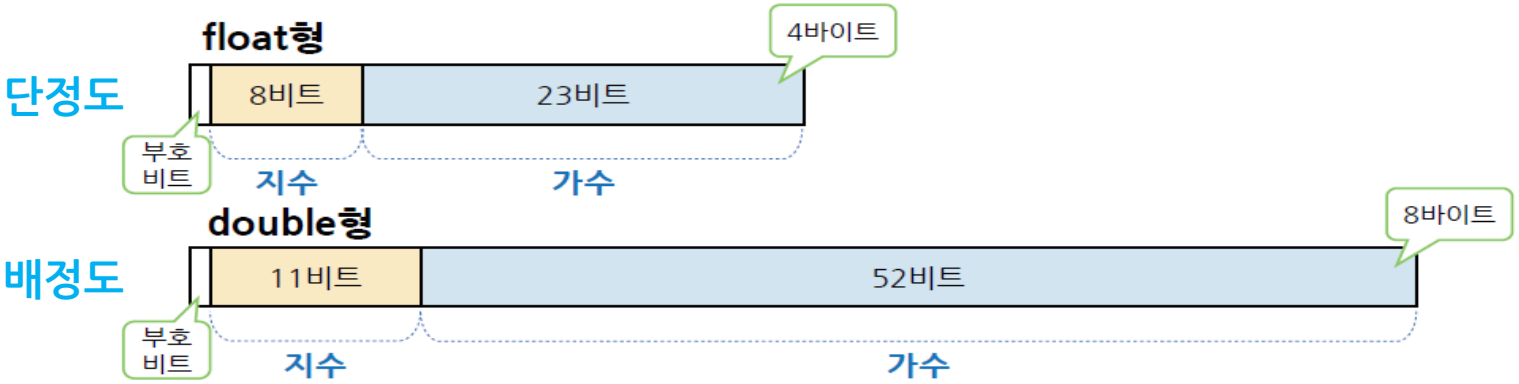
❖실수의 표현 방식

- 고정소수점(fixed point) vs 부동소수점(floating point)
- 부동소수점 방식을 주로 사용



❖실수의 정밀도

- 단정도(float형)와 배정도(double형)



자료형_실수형

❖float 형과 double 형의 정밀도 비교

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      float  rate1 = 9.876543210987654;
06      double rate2 = 9.876543210987654;
07
08      printf("float  rate1 = %.15f\n", rate1);
09      printf("double rate2 = %.15f\n", rate2);
10  }
```

실행 결과

```
float  rate1 = 9.876543045043945
double rate2 = 9.876543210987654
```

float형은 소수점 이하 6자리까지 올바르게 출력

소수점 이하 15자리 출력

자료형_실수형 :유효범위

- 실수형의 오버플로우
  - 표현 가능한 최대값보다 큰 값 저장 시 → INF로 설정
- 실수형의 언더플로우
  - 표현 가능한 최소값보다 작은 값을 저장 시 → 가수 부분을 줄이고 지수 부분을 늘려서 실수를 표현하거나, 만일 그것이 불가능해지면 0으로 만들어 버린다.

데이터형	크기	유효 범위
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	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{308}$

자료형_실수형 :오버플로우와 언더플로우(1/2)

```
01  #include <stdio.h>
02
03  int main(void)
04  {
05      float num = 3.40282e38;
06      printf("num = %.15e\n", num);
07
08      num = 3.40282e40;
09      printf("num = %.15e\n", num);
10
11      num = 1.17549e-38;
12      printf("num = %.15e\n", num);
13
```

float형의 최대값 저장

오버플로우 발생 → INF로 설정

float형의 최소값 저장

지수 표시 방식으로 소수점 이하 15자리까지 출력

자료형_실수형 :오버플로우와 언더플로우(2/2)

```
14        num = 1.17549e-42;
15        printf("num = %.15e\n", num);
16
17        num = 1.17549e-46;
18        printf("num = %.15e\n", num);
19    }
```

가수부를 줄여서 실수 표현

언더플로우 발생 → 0으로 설정

실행 결과

```
num = 3.402820018375656e+38
num = inf
num = 1.175490006797048e-38
num = 1.175689411568522e-42
num = 0.000000000000000e+00
```

오버플로우 발생

언더플로우 발생

자료형_정수형

❖자료형의 유효범위

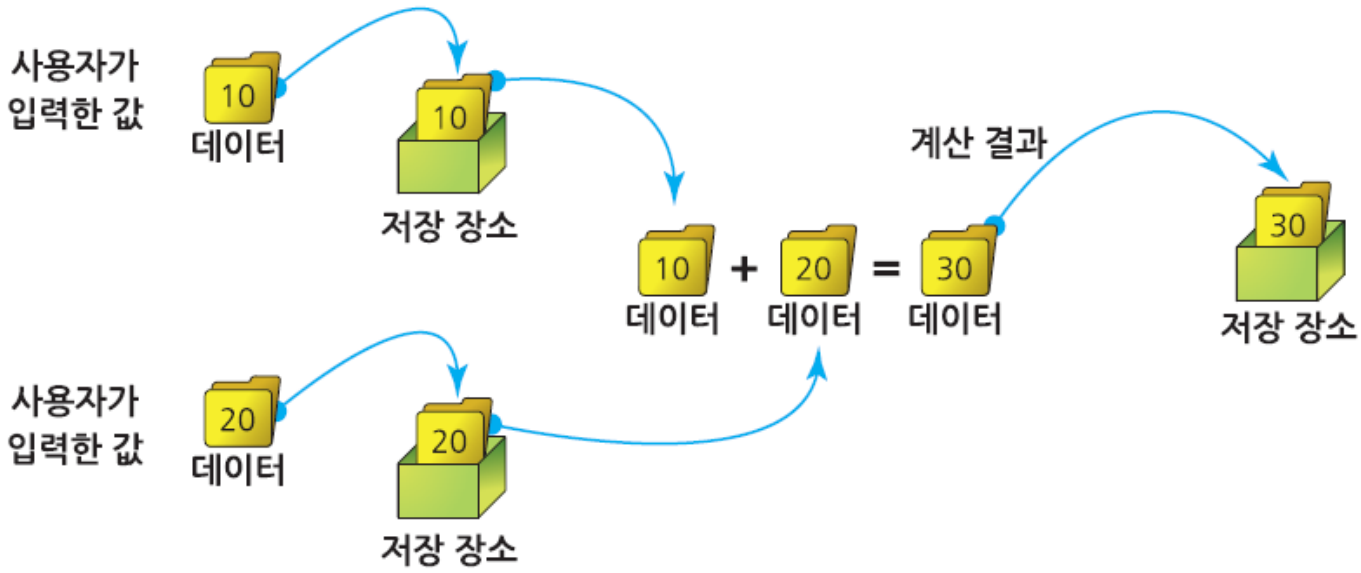
분류	데이터형	바이트 크기	유효 범위
문자형	char	1	$-128(-2^7) \sim 127(2^7-1)$
	unsigned char	1	$0 \sim 255(2^8-1)$
정수형	short	2	$-32768(-2^{15}) \sim 32767(2^{15}-1)$
	unsigned short	2	$0 \sim 65535(2^{16}-1)$
	int	4	$-2147483648(-2^{31}) \sim 2147483647(2^{31}-1)$
	unsigned int	4	$0 \sim 4294967295(2^{32}-1)$
	long	4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

변수와 상수

- 프로그램에서 사용되는 데이터에는 변수와 상수가 있다.
  - 변수 : 값을 **변경할 수 있는** 데이터
  - 상수 : 값을 **변경할 수 없는** 데이터

변수와 상수_변수의 필요성(1/2)

❖데이터를 보관해둘 필요가 있을 때, 변수를 사용한다.



변수와 상수_변수의 필요성(2/2)

❖변수를 이용하면 '어떤 값이 될지 모르는 값을 처리하는 프로그램'을 작성할 수 있다.

변수를 사용하지 않는 경우

InfinityWarPlayer.exe

```
int main(void)
{
  "InfinitWar.mp4" 파일을 연다.
  동영상을 재생한다.
  파일을 닫는다.

  return 0;
}
```

BlackPantherPlayer.exe

```
int main(void)
{
  "BlackPanther.mp4" 파일을 연다.
  동영상을 재생한다.
  파일을 닫는다.

  return 0;
}
```

파일 이름이 달라지면 프로그램을 다시 만든다.

⋮

같은 일을 하는 프로그램을 매번 다시 만들어야 하므로 비효율적이다.

변수를 사용하는 경우

MoviePlayer.exe

```
int main(void)
{
  char filename[80]; // 변수 사용
  filename에 동영상 파일 이름을 입력받는다.
  filename 파일을 연다.
  동영상을 재생한다.
  파일을 닫는다.

  return 0;
}
```

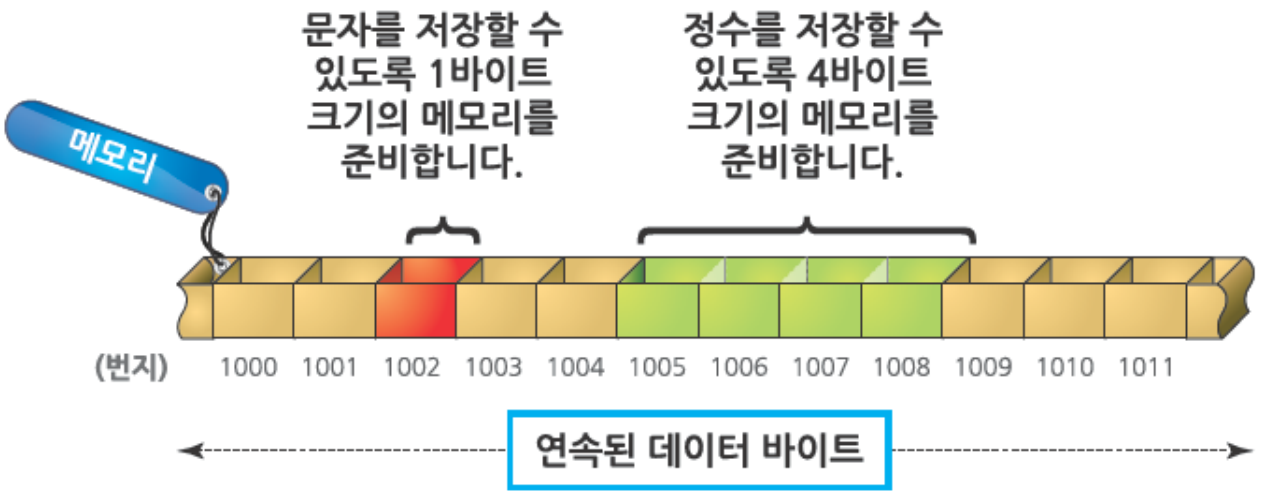
입력된 파일 이름을 저장하는 변수를 이용한다.

한 프로그램으로 여러 가지 동영상을 재생할 수 있다.

변수와 상수_변수

- 변경될 수 있는 데이터는 메모리(memory)에 보관한다.
  - 메모리는 연속된 데이터 바이트의 모임을 말하며, 메모리의 각 바이트는 주소를 갖는다.

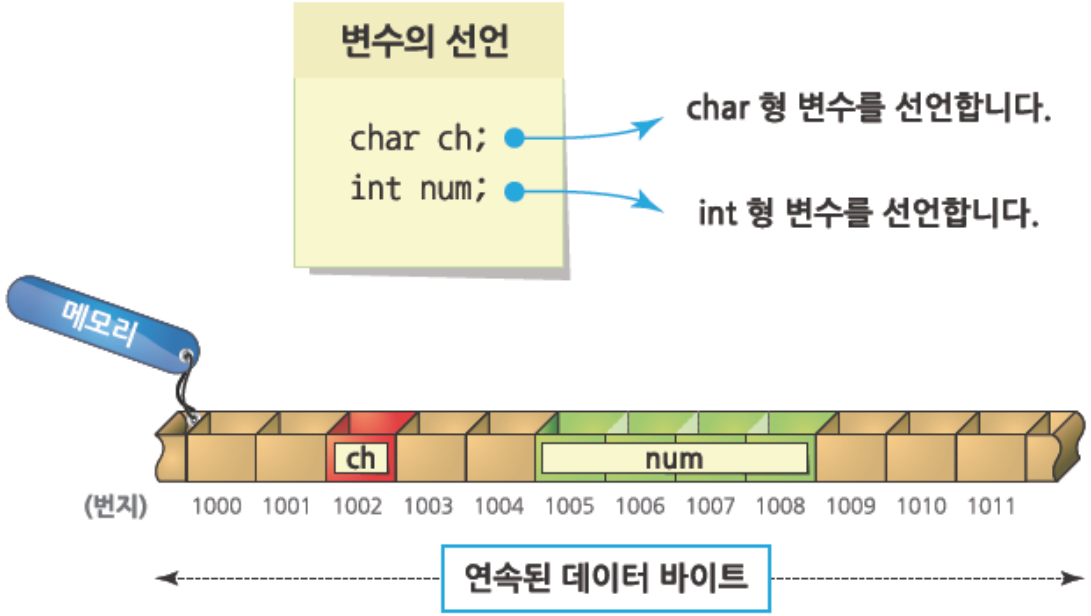
- 메모리에 저장할 데이터 값의 형식(type)에 따라 메모리가 얼마만큼 필요한지가 결정된다.



변수와 상수_변수

❖변수의 선언

- 변수를 사용하기 위해서 특정 크기의 메모리를 준비하는 것을 "변수를 메모리에 할당한다"고 한다.
  - 컴파일러에게 변수의 데이터형과 변수 이름을 알려주는 것
- 저장될 값의 데이터형에 따라 필요한 만큼 메모리를 할당



변수와 상수_변수

❖변수 선언의 기본적인 형식

- 변수를 선언하려면 변수의 데이터형과 이름이 필요하다.

형식

```
데이터형 변수명;
데이터형 변수명1, 변수명2, … ;
```

사용예

```
int length;
double height, weight;
unsigned char red, green, blue;
```

- 같은 형의 변수를 여러 개 선언하려면 콤마(,) 다음에 변수 이름 나열

```
double height, weight;
```

변수와 상수_변수

- **식별자(identifier)** : 변수 이름, 함수 이름처럼 프로그래머가 만들어서 사용하는 이름
- **식별자를 만드는 규칙**
  - 반드시 영문자, 숫자, 밑줄 기호(_)만을 사용해야 한다.
  - 첫 글자는 반드시 영문자 또는 밑줄 기호(_)로 시작해야 한다.
  - 밑줄 기호(_)를 제외한 다른 기호를 사용할 수 없다.
  - 대소문자를 구분해서 만들어야 한다. red, Red, RED은 모두 다른 이름이다.
  - C 언어의 키워드는 식별자로 사용할 수 없다.

◈ C의 키워드

auto	break	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			

변수와 상수_변수

- 변수의 선언-의미가 명확한 이름을 사용하는 것이 좋다.
  - 올바른 변수 선언의 예

```
int usage2019;
double _amount;
double tax_rate;
double taxRate;
```

변수 이름은 _로 시작 가능

여러 단어를 연결할 때는 _ 사용

연결되는 단어의 첫 글자를 대문자로 지정

  - 잘못된 변수 선언의 예

```
long annual-salary;
int total amount;
int 2019income;
char case;
```

변수 이름에 - 기호를 사용할 수 없다.

변수 이름에 빈칸을 포함할 수 없다.

변수 이름은 숫자로 시작할 수 없다.

키워드는 변수 이름으로 사용할 수 없다.

변수와 상수_변수

❖변수 선언문의 위치

ANSI C

```
int main(void)
{
   int num;
   float sum;

   scanf("%d", &num);
   sum = num * (num + 1) * 0.5;
   printf("sum = %.1f\n", sum);

   return 0;
}
```

변수 선언문이 함수 시작 부분에 모여 있어야 한다.

C99

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

   float sum;
   sum = num * (num + 1) * 0.5;
   printf("sum = %.1f\n", sum);

   return 0;
}
```

변수를 필요한 곳에서 선언할 수 있다.

변수와 상수_변수

❖변수의 초기화(1/2)

- 변수가 처음 메모리에 할당될 때 값을 지정하는 것

형식

데이터형 변수명 = 초기값;
데이터형 변수명1 = 초기값1, 변수명2 = 초기값2, … ;

사용예

```
double tax_rate = 0.2;
char gender = 'F';
int price = 0, total_price = 0;
```

- 변수의 데이터형과 초기값의 데이터형이 일치하지 않으면 데이터형에 맞춰서 값을 변환해서 초기화한다.

```
float rate = 0.2;
int weight = 50.5;
```

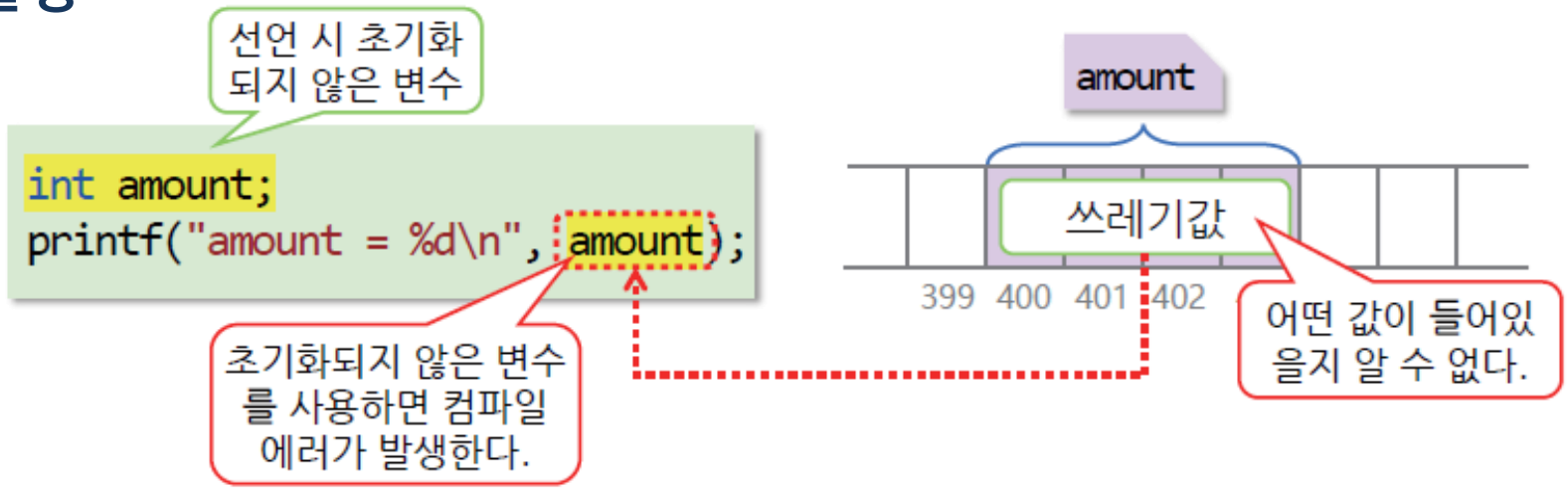
0.2은 double형이므로 float형으로 변환하면서 컴파일 경고 발생

50.5은 double형이므로 int형으로 변환하면서 컴파일 경고 발생

변수와 상수_변수

❖변수의 초기화(2/2)

- 변수를 선언할 때 따로 **초기화를 하지 않으면 쓰레기 값**을 갖는다.
- 초기화되지 않은 변수를 사용하는 것은 위험하다. → **컴파일 경고** 발생



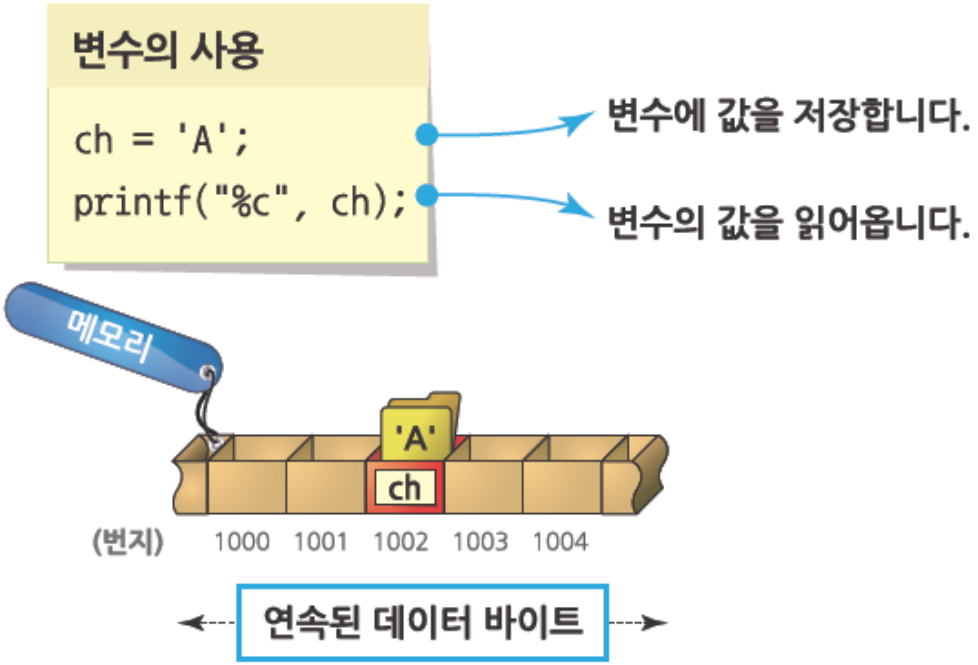
- 변수가 어떤 초기값을 가져야 하는지 알 수 없으면 0으로 초기화한다.

```
int amount = 0;     // 어떤 값이 될지 아직 알 수 없으면 0으로 초기화한다.
float sum = 0;      // 어떤 값이 될지 아직 알 수 없으면 0으로 초기화한다.
```

변수와 상수_변수

❖변수의 사용

- 변수를 선언하고 나면 변수의 이름을 이용해서 메모리에 접근할 수 있다.
- 특정 값을 변수에 저장하거나 메모리에 보관된 변수의 값을 읽어올 수 있다.



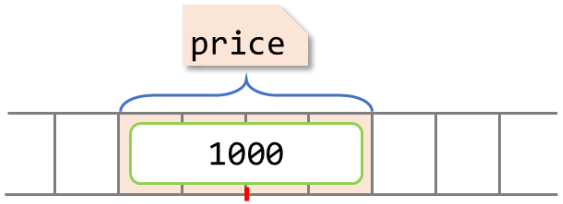
변수와 상수_변수

❖변수의 사용(1/2)

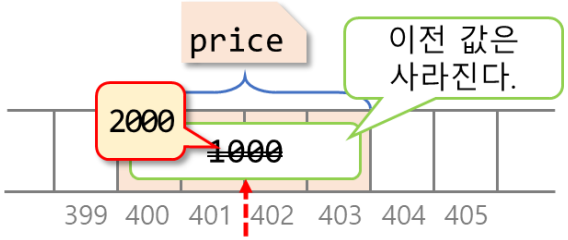
- 변수의 값을 읽어오거나 저장하려면 변수명을 이용한다.
- 변수에 값을 저장하려면 **대입연산자(=)**를 사용한다.

```
int price;
price = 1000;
printf("price = %d\n", price);
```

price에 저장된 값인 1000을 읽어온다.




```
int price;
price = 1000;
printf("price = %d\n", price);
price = 2000 ;
```



변수와 상수_변수

❖변수의 사용(2/2)

- **변수의 대입(assignment)**
  - 변수에 값을 저장하는 것

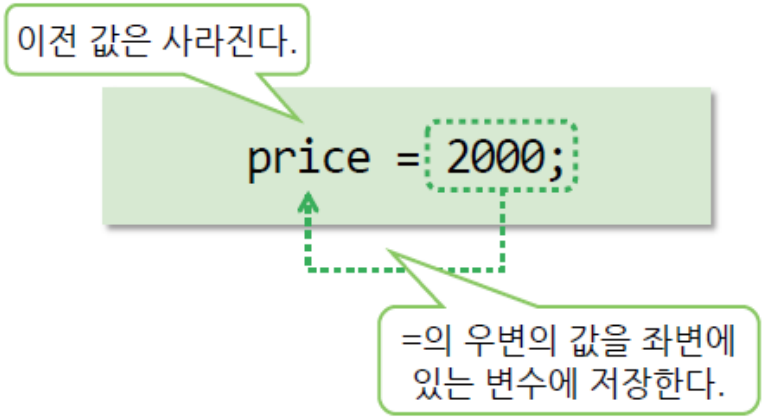
형식

```
변수명 = 값;
변수명 = 수식;
```

사용예

```
tax_rate = 0.1;
price = 2000;
total_price = amount * price;
```

- **변수의 데이터형과 같은 형의 값을 대입해야 한다.**
  - 데이터형이 같지 않으면 값을 변수의 데이터형으로 변환해서 저장




```
double weight;
weight = 50;
```

50을 50.0으로 변환해서 대입

```
int length;
length = 12.5;
```

12.5를 12로 변환하면서 데이터 손실 → 컴파일 경고 발생

변수와 상수_변수

❖변수의 선언과 사용

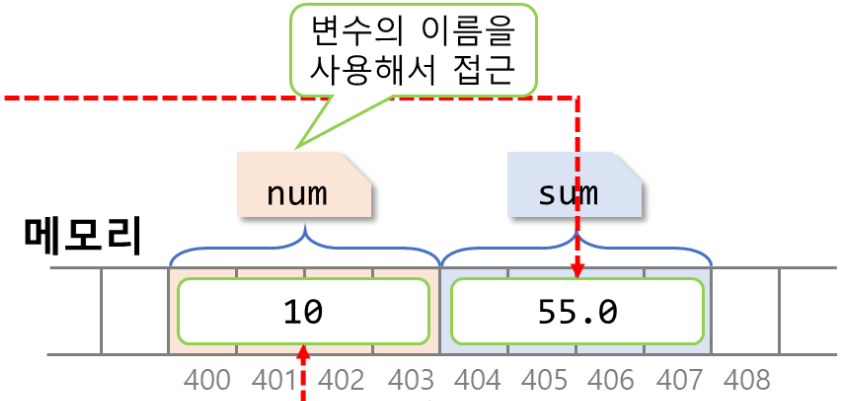
```
int main(void)
{
  int num;
  float sum;

  scanf("%d", &num);
  sum = num * (num + 1) * 0.5
  printf("sum = %.1f\n", sum);

  return 0;
}
```

변수가 사용할 메모리 할당

입력받은 값을 변수에 저장

연산의 결과를 변수에 저장

변수의 이름을 사용해서 접근



변수의 주소는 알 필요 없다.

[실행 결과]

```
10
sum = 55.0
```

변수와 상수_변수

❖변수의 초기화 및 변수의 사용 방법

```
03  int main(void)
04  {
05      int amount;                 // 수량 → 초기화하지 않은 경우
06      int price = 0;              // 단가 → 정수형 변수는 0으로 초기화
07      int total_price = 0;        // 합계 금액 → 정수형 변수는 0으로 초기화
08
09      printf("amount = %d, price = %d\n", amount, price);
10
11      printf("수량? ");
12      scanf("%d", &amount);
13
14      price = 2000;
15
16      total_price = amount * price;    // 합계 금액
17      printf("합계: %d원\n", total_price);
18
19      return 0;
20  }
```

초기화되지 않은 변수

초기화되지 않은 변수 사용 시 컴파일 경고 발생

변수의 대입

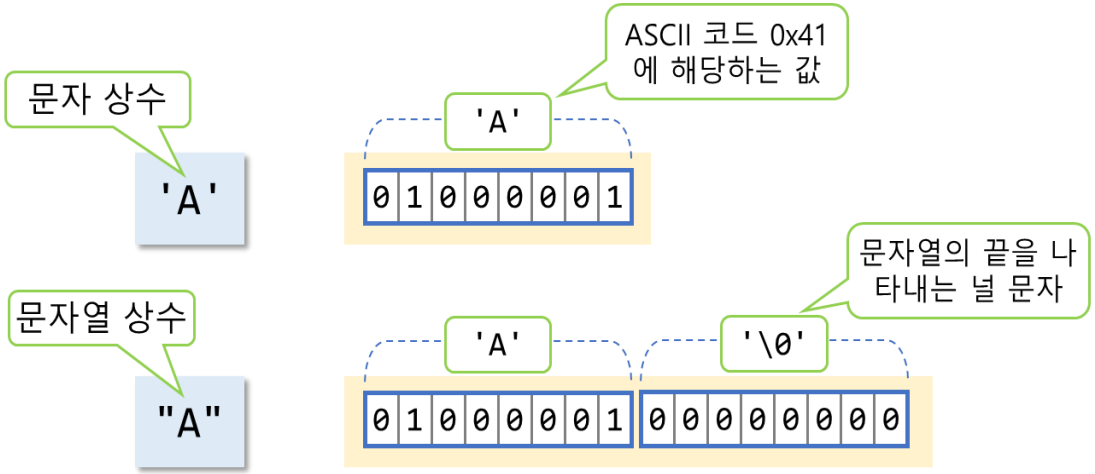
실행결과

```
amount = -858993460, price = 0
수량? 2
합계: 4000원
```

쓰레기 값

변수와 상수_상수

- 프로그램에서 값이 변경되지 않는 요소
- 임시값(temporary value)
  - 값이 메모리에 저장되지 않고, 한 번만 사용된 다음 없어져 버림
- 상수의 종류
  - 리터럴 상수 : 소스 코드에서 직접 사용되는 값 자체
    - 문자 상수 : 'A', '₩xa'
    - 정수형 상수 : 0x12, 123u
    - 실수형 상수 : 12.34, 0.5F
    - 문자열 상수 : "hello", "A"
  - 기호 상수



변수와 상수_리터럴 상수

상수의 종류	구분	예	데이터형
문자형	일반 문자	‘x’, ‘y’	int
	이스케이프 시퀀스	‘₩b’, ‘₩t’, ‘₩xa’	int
정수형	10진수 정수	-10, 10	int
	16진수 정수	0xa, 0XA	int
	8진수 정수	012	int
	unsigned형 정수	65536u, 65536U	unsigned int
	long형 정수	1234567l, 1234567L	long
	unsigned long형 정수	1234567ul, 1234567UL	unsigned long
실수형	부동소수점 표기 실수	56.78, .5	double
	지수 표기 실수	5.678e1, .5e0	double
	float형 실수	0.25f, 0.25F	float
문자열	문자열 상수	“apple”, “x”	char[ ]

변수와 상수_리터럴 상수

❖리터럴 상수의 크기

```
03  int main(void)
04  {
05      printf("sizeof(\'a\') = %d\n", sizeof('a'));     // 4바이트
06      printf("sizeof(12345) = %d\n", sizeof(12345));
07      printf("sizeof(12345U) = %d\n", sizeof(12345U));
08      printf("sizeof(12345L) = %d\n", sizeof(12345L));
09
10      printf("sizeof(12.34F) = %d\n", sizeof(12.34F));
11      printf("sizeof(12.34567) = %d\n", sizeof(12.34567));
12      printf("sizeof(1.234e-5) = %d\n", sizeof(1.234e-5));
13
14      printf("sizeof(\"abcde\") = %d\n", sizeof("abcde"));
15
16      return 0;
```

작은따옴표 출력 시 \'로 표기

문자 상수는 int형으로 간주

큰따옴표 출력 시 \"로 표기

"abcde"를 저장하는데 필요한 배열의 크기

실행결과

```
sizeof('a') = 4
sizeof(12345) = 4
sizeof(12345U) = 4
sizeof(12345L) = 4
sizeof(12.34F) = 4
sizeof(12.34567) = 8
sizeof(1.234e-5) = 8
sizeof("abcde") = 6
```

변수와 상수_매크로 상수

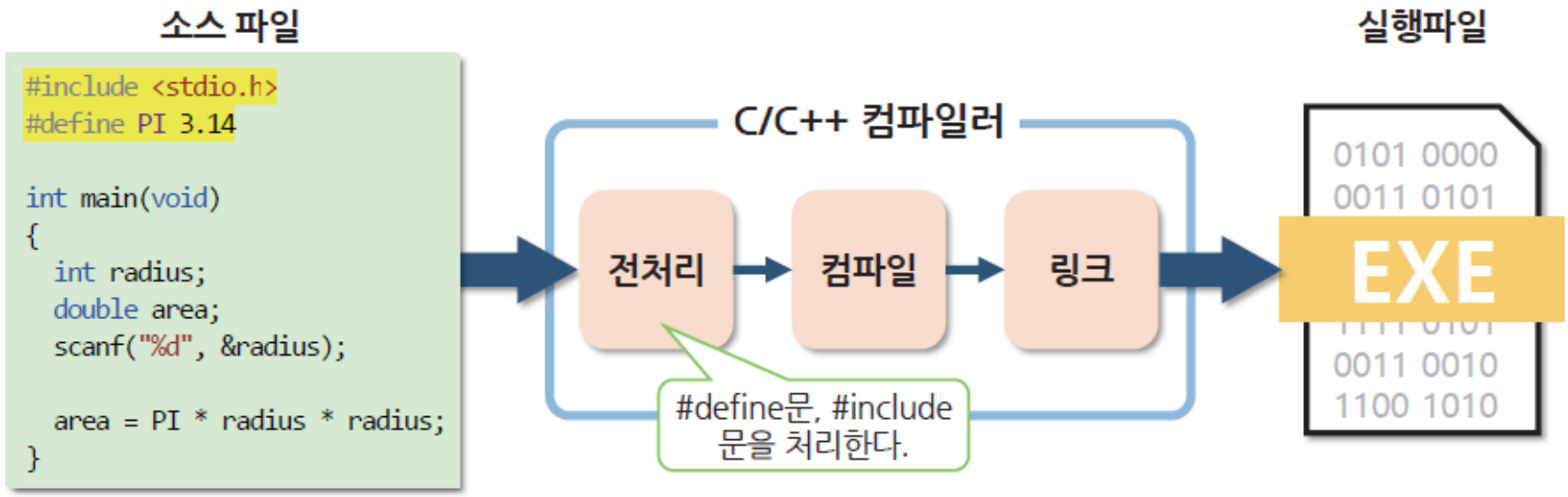
❖매크로 상수

- **#define문으로 정의되는 상수**
  - 전처리기가 처리하는 문장
  - 전처리기는 컴파일러가 소스 파일을 컴파일하기 전에 먼저 수행되며, 프로그래머가 작성한 소스 파일을 컴파일할 수 있도록 변환해서 준비함.

형식 `#define 매크로명 값`

사용예

```
#define BUF_SIZE 256
#define SEPARATOR '\n'
#define PI 3.141592
#define GREETING "hello"
```



변수와 상수_매크로 상수

❖전처리기(preprocessor)의 #define 처리

- 전처리기는 매크로 상수를 특정 값으로 대치(replace)한다.

전처리기 수행 전

```
#define PI 3.14

int main(void)
{
  double r;
  double area

  scanf("%lf", &r);
  area = PI * r * r;
  printf("PI = %.2f\n", PI);
}
```

전처리기 수행 후

```
int main(void)
{
  double r;
  double area;

  scanf("%lf
  area = 3.14 * r * r;
  printf("PI = %.2f\n", 3.14);
}
```

#define문은 사라진다.

매크로 대치

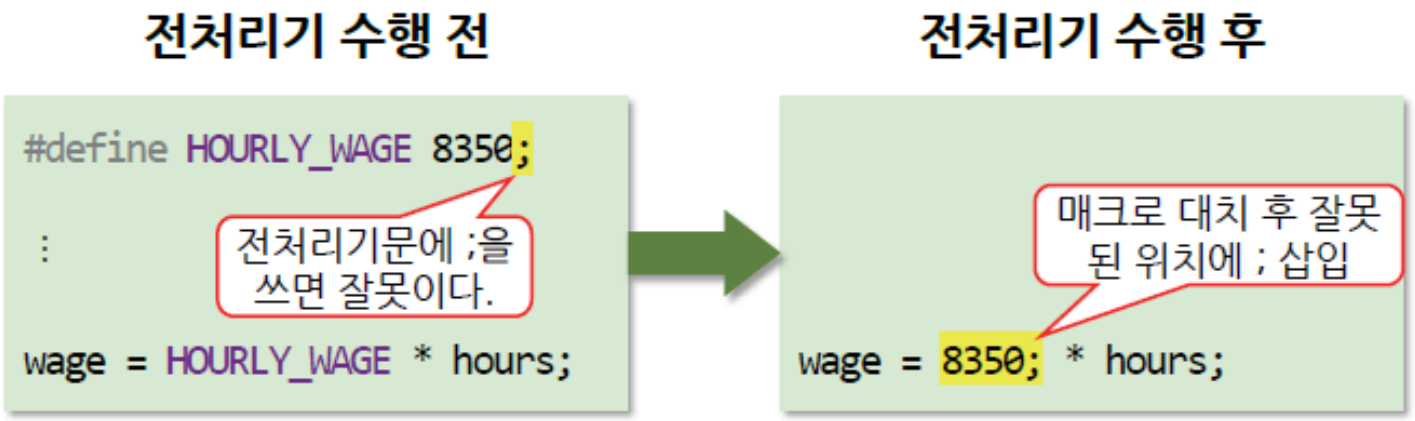
매크로 대치

문자열의 일부는 대치되지 않는다.

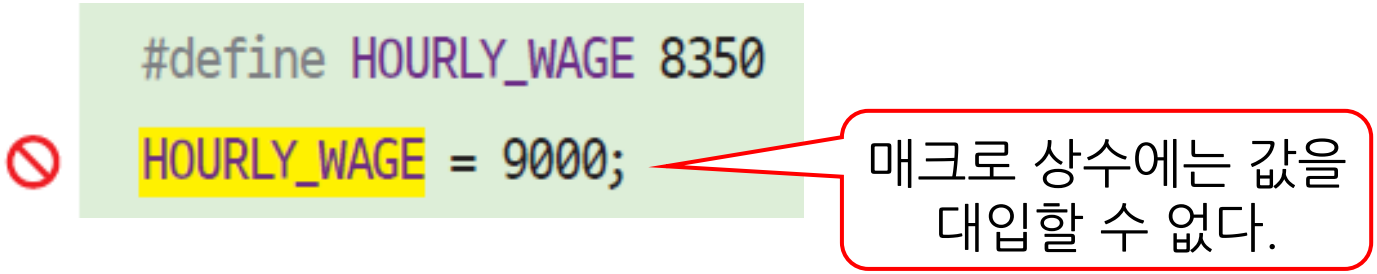
변수와 상수_매크로 상수

❖전처리기(preprocessor)의 #define 처리

- #define문의 끝에는 세미콜론(;)이 필요 없다.



- 매크로 상수의 값은 변경할 수 없다.



변수와 상수_매크로 상수

❖매크로 상수의 정의 및 사용

```
01  #include <stdio.h>
02  #define PI 3.14   // 매크로 상수 정의
03  int main(void)
04  {
05      double radius = 0;
06      double area = 0;
07
08      printf("반지름? ");
09      scanf("%lf", &radius);  // double형 입력
10
11      area = PI * radius * radius;
12      printf("원의 면적: %.2f\n", area);
13      printf("PI = %.2f\n", PI);
14      return 0;
15  }
16
```

3.14로 대치

3.14로 대치

문자열이므로 대치되지 않는다.

실행결과

```
반지름? 5
원의 면적: 78.50
PI = 3.14
```

변수와 상수_const 변수

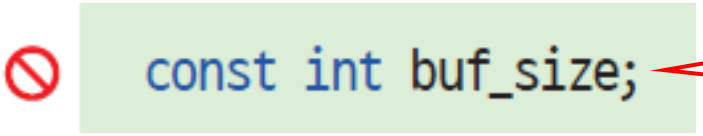
❖const 변수-값을 변경할 수 없는 변수

- 변수 선언 시 데이터형 앞에 const를 지정하면 값을 변경할 수 없는 변수가 된다.

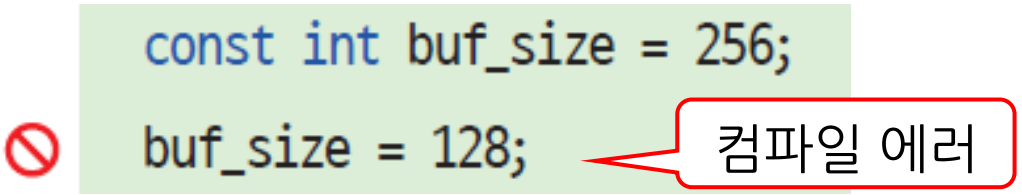
형식 `Const 데이터형 변수명 = 초기값;`

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

- **const 변수는 선언시 반드시 초기화해야 한다.**



- const 변수의 값을 변경하려고 하면 컴파일 에러가 발생한다.



변수와 상수_const 변수

❖const 변수의 선언 및 사용

```
03  int main(void)
04  {
05      int amount = 0, price = 0;
06      const double VAT_RATE = 0.1; // 부가가치세율
07      int total_price = 0;
08
09      printf("수량? ");
10      scanf("%d", &amount);
11
12      printf("단가? ");
13      scanf("%d", &price);
14
15      total_price = amount * price * (1 + VAT_RATE);
16      printf("합계: %d원\n", total_price);
```

const 변수 선언

const 변수 사용

실행결과

```
수량? 2
단가? 5000
합계: 11000원
```

변수와 상수_기호 상수

- 이름이 있는 상수를 **기호 상수**라고 한다.
- 기호 상수의 장점
  - 이름이 있는 상수를 사용하면 프로그램이 수정하기 쉬워진다.

리터럴 상수를 사용하는 경우

```
double area, perimeter;
int radius = 5;

area = 3.14 * radius * radius;
perimeter = 2 * 3.14 * radius;
```

3.141592

3.141592

리터럴 상수를 사용하는
모든 곳을 수정해야 한다.

기호 상수를 사용하는 경우

```
#define PI 3.14
double area, perimeter;
int radius = 5;

area = PI * radius * radius;
perimeter = 2 * PI * radius;
```

3.141592

PI를 사용하는 곳은
수정할 필요 없다.

기호 상수를 정의하는 곳만
수정하면 된다.

변수와 상수_기호 상수

- 이름이 있는 상수를 사용하면 프로그램의 가독성(readability)이 향상된다.(이해하기 쉽다.)

리터럴 상수를 사용하는 경우

```
int amount, price, total;
scanf("%d %d", &amount, &price);

total = amount * price * (1 + 0.1);
```

0.1이 어떤 의미인지
알 수 없다.

기호 상수를 사용하는 경우

```
const double vat_rate = 0.1;

int amount, price, total;
scanf("%d %d", &amount, &price);

total = amount * price * (1 + vat_rate);
```

vat_rate를 사용하면
의미를 명확히 알 수 있다.

학습정리

❖변수와 상수

- 변수 : 값이 변경될 수 있는 데이터, 값을 보관하기 위해서 메모리 사용
- 변수의 선언 : 데이터형과 변수명이 필요

```
int grade;
```

- 변수의 사용 : 변수의 이름을 이용
- 변수의 초기화 : 변수를 초기화하지 않으면 쓰레기 값을 가지므로, 초기화하는 것이 좋다.

```
int grade = 100;
```

- 상수 : 프로그램에서 값이 변경되지 않는 요소, 리터럴 상수와 매크로 상수, const 변수가 있다.

'A', 10, 3.14 → 리터럴 상수

```
#define MAX 100 → 매크로 상수
const int max = 100; → const 변수
```

학습정리

❖자료형

- 문자형 : char형이 있으며, 문자 코드를 저장한다.
- 정수형 : short, int, long형이 있으며, 부호 있는 정수(signed)와 부호 없는 정수(unsigned)로 나누어진다.
- 실수형 : float형과 double형이 있다.
- sizeof 연산자 : 데이터형이나 변수의 바이트 크기를 구한다.

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