3장. 데이터형과 변수

박 종 혁 교수
서울과학기술대학교 컴퓨터공학과
UCS Lab
Tel: 970-6702
Email: jhpark1@seoultech.ac.kr

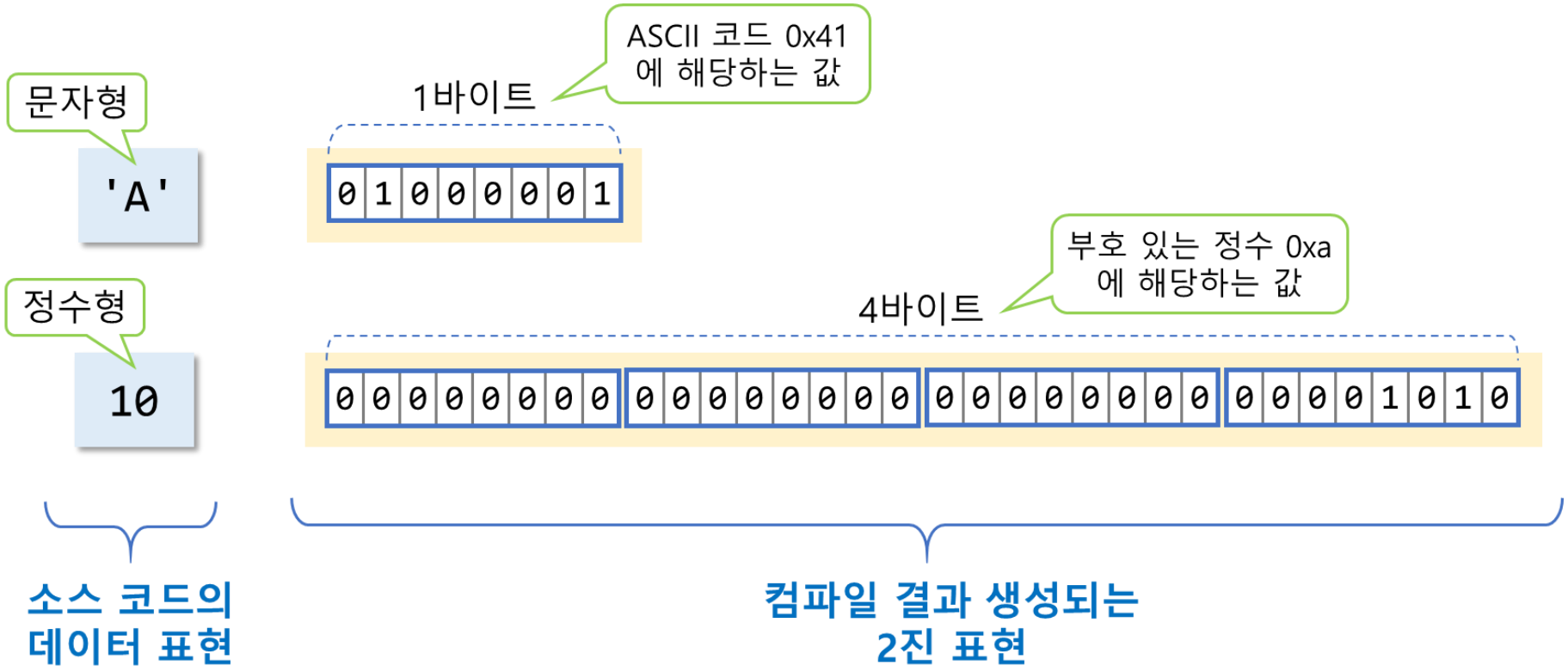
목차

❖데이터형
- 데이터형의 기본 개념
- 정수형
- 문자형
- 실수형

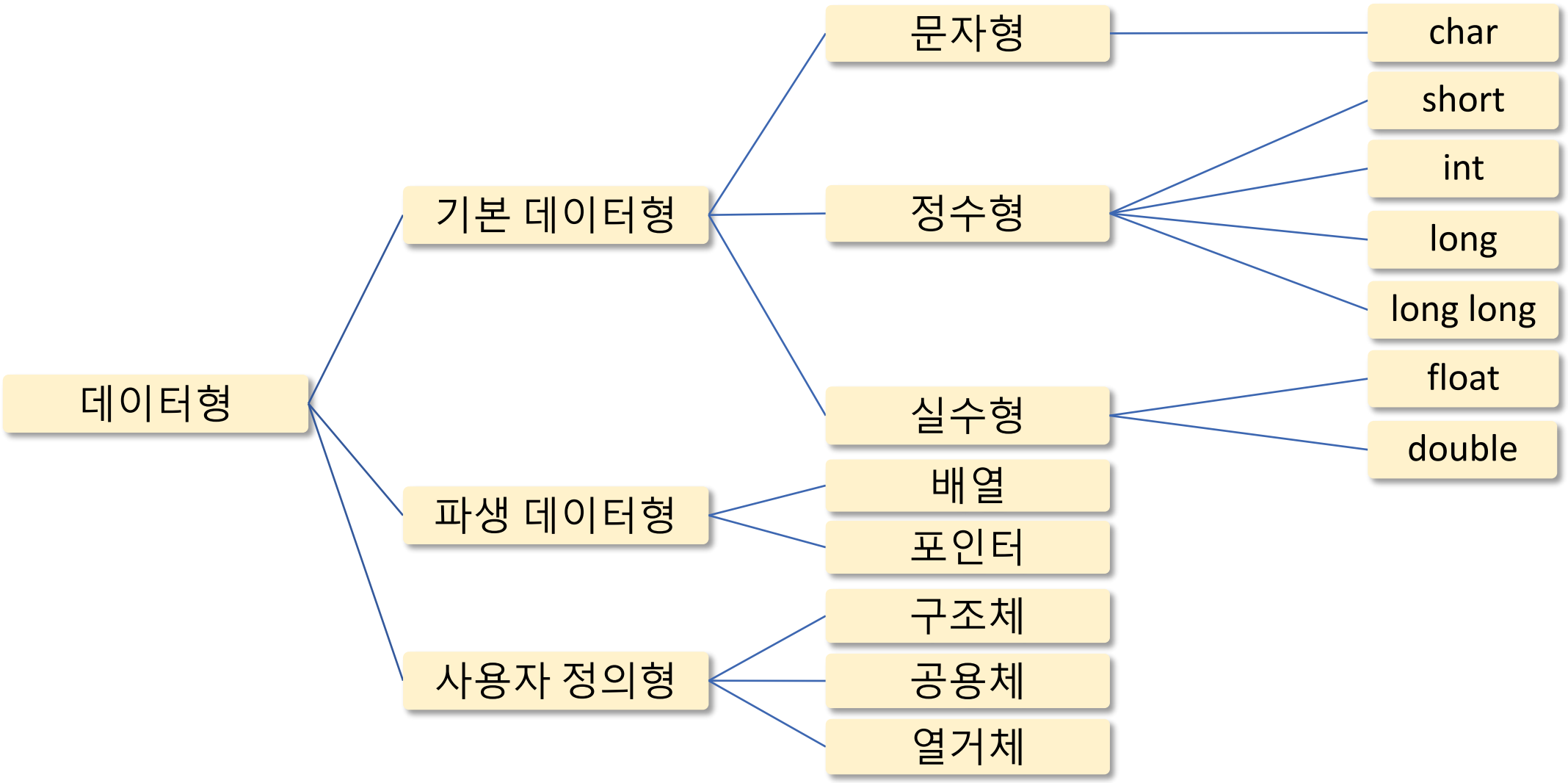
❖변수와 상수
- 변수
- 상수

데이터의 2진 표현

- 컴퓨터 시스템에서 2진 데이터로 값을 표현하고 저장하는 방식
- 컴파일러는 데이터형에 따라 값을 저장하는데 필요한 **메모리의 크기와 2진 표현을 결정**한다



기본 데이터형


데이터형
기본 데이터형
파생 데이터형
사용자 정의형
문자형
정수형
실수형
배열
포인터
구조체
공용체
열거체
char
short
int
long
long long
float
double

기본형의 기능별 분류

- 정수적형

```
char            signed char   unsigned char
short           int           long
unsigned short  unsigned      unsigned long
```

- 부동형

```
float           double        long double
```

- 산술형

  정수적형 + 부동형

sizeof 연산자

- 데이터형의 크기는 플랫폼에 따라 다르다
- 데이터형이나 값의 바이트 크기를 구하려면 sizeof 연산자를 이용한다

형식	sizeof(데이터형) sizeof(값) sizeof 값
사용예	

```
sizeof(char)
sizeof(int)

sizeof(num)
sizeof(num + 1)

sizeof 3.141592
```

- 소스 코드에서 데이터형이나 값의 크기가 필요할 때는 sizeof 연산자로 구한 크기를 사용하는 것이 좋다

```
size1 = 4 * 10;              // 의미가 불분명한 코드
size2 = sizeof(float) * 10;  // 의미가 명확한 코드
```

예제 3-1 : 데이터형의 크기 구하기

```c
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));
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  }
```

여러 가지 데이터형의 변수 선언

데이터형의 바이트 크기 구하기

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

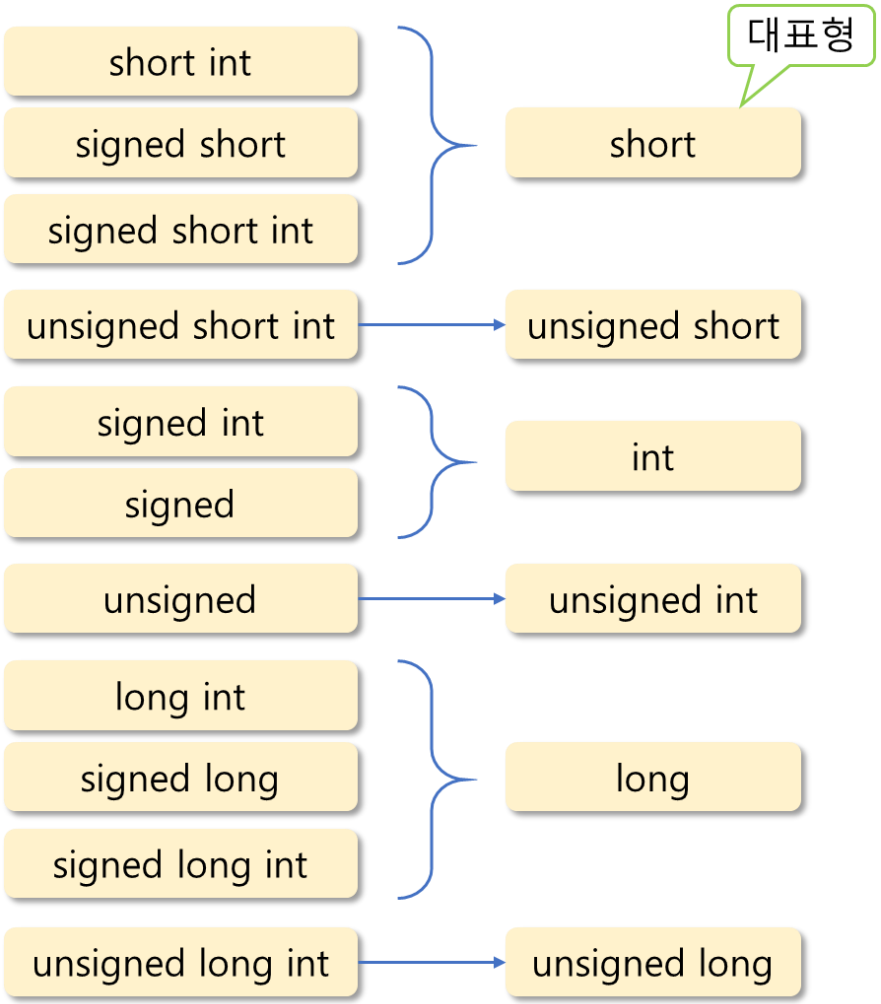
정수형

❖프로그래머가 용도에 따라 적절한 데이터형을 선택할 수 있도록 다양한 크기의 정수형을 제공한다

- sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long)

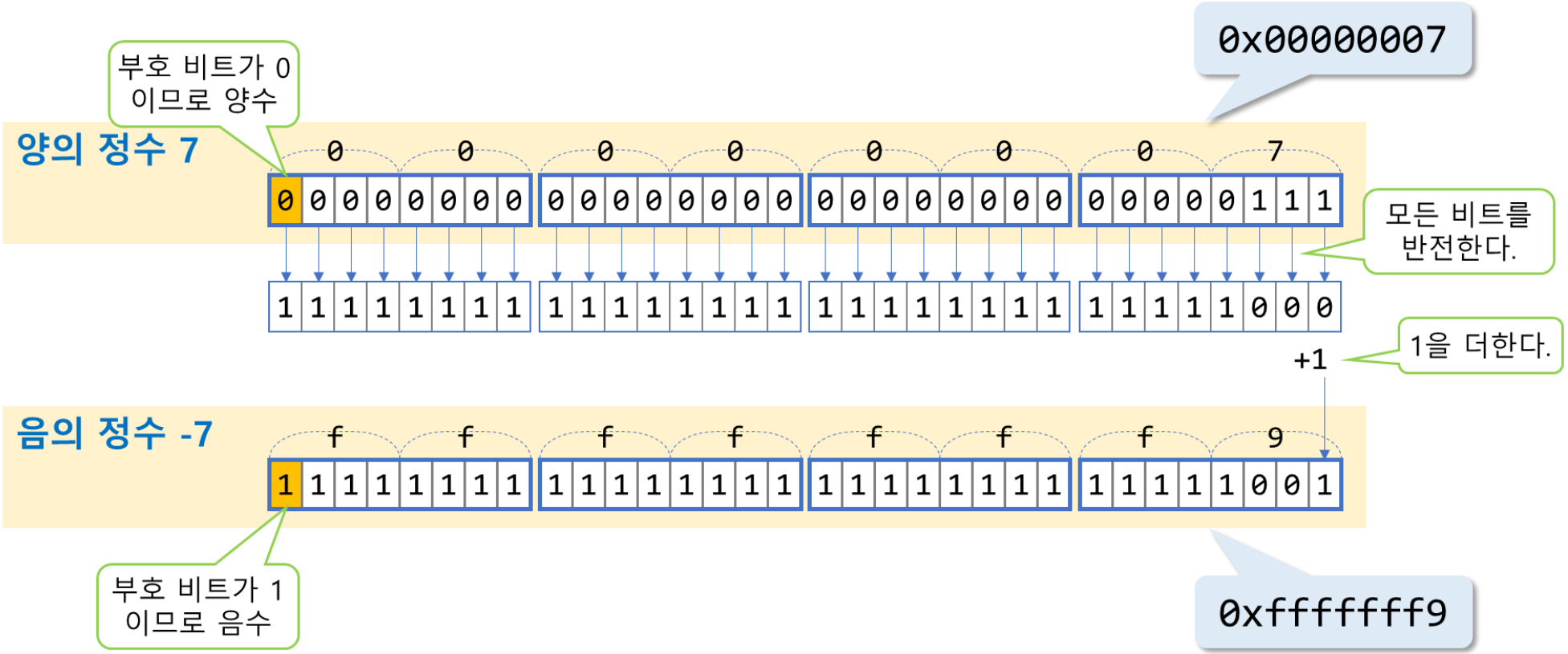
❖부호 있는 정수형과 부호 없는 정수형

- signed는 생략 가능
- unsigned : 부호 없는 정수형



정수의 2진 표현

- 부호 있는 정수형은 최상위 비트를 **부호 비트**로 사용
- 음수를 나타내기 위해 **2의 보수**를 사용



예제 3-2 : 정수의 2진 표현

```c
#include <stdio.h>

int main(void)
{
    printf(" 7 = %08x\n", 7);
    printf("-7 = %08x\n", -7);
    printf("7+(-10) =  %08x\n", 7 + (-10));

    return 0;
}
```

%08x는 8문자 폭에 맞춰서 16진수로 출력하면서, 빈칸에는 0을 출력한다.

실행결과

```
 7 = 00000007
-7 = fffffff9
7+(-10) = fffffffd
```

2의 보수로 표현된 음의 정수

부호 없는 정수형

- 최상위 비트를 값을 저장하는 용도로 사용

```
short a = -10;
```

부호 비트가 1 이므로 음수

f	f	f	6
1 1 1 1 1 1 1 1	1 1 1 1 0 1 1 0		

```
unsigned short b = 65526;
```

2^{15}에 해당하는 값으로 간주

f	f	f	6
1 1 1 1 1 1 1 1	1 1 1 1 0 1 1 0		

2진 표현은 같지만 의미가 다르다.

예제 3-3 : 부호 있는 정수와 부호 없는 정수

```
03  int main(void)
04  {
05      short a = -10;
06      unsigned short b = 65526;
07
08      printf("a = %d, %08x\n", a, a);
09      printf("b = %u, %08x\n", b, b);
10
11      return 0;
12  }
```

a, b의 값을 10진수와 16진수로 출력해서 2진 표현을 비교해본다.

부호 없는 정수를 출력할 때는 %u를 사용한다.

실행결과

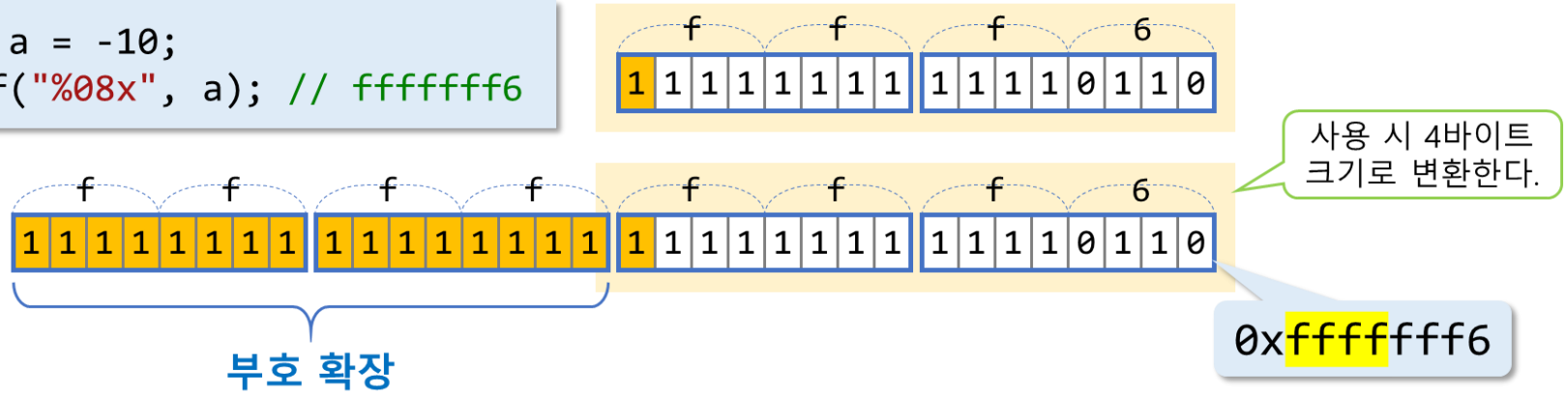
```
a = -10, fffffff6
b = 65526, 0000fff6
```

printf 함수는 2바이트 크기의 short형 변수의 값을 출력할 때 4바이트 크기로 변환하므로 하위 2바이트 값만 비교해야 한다.

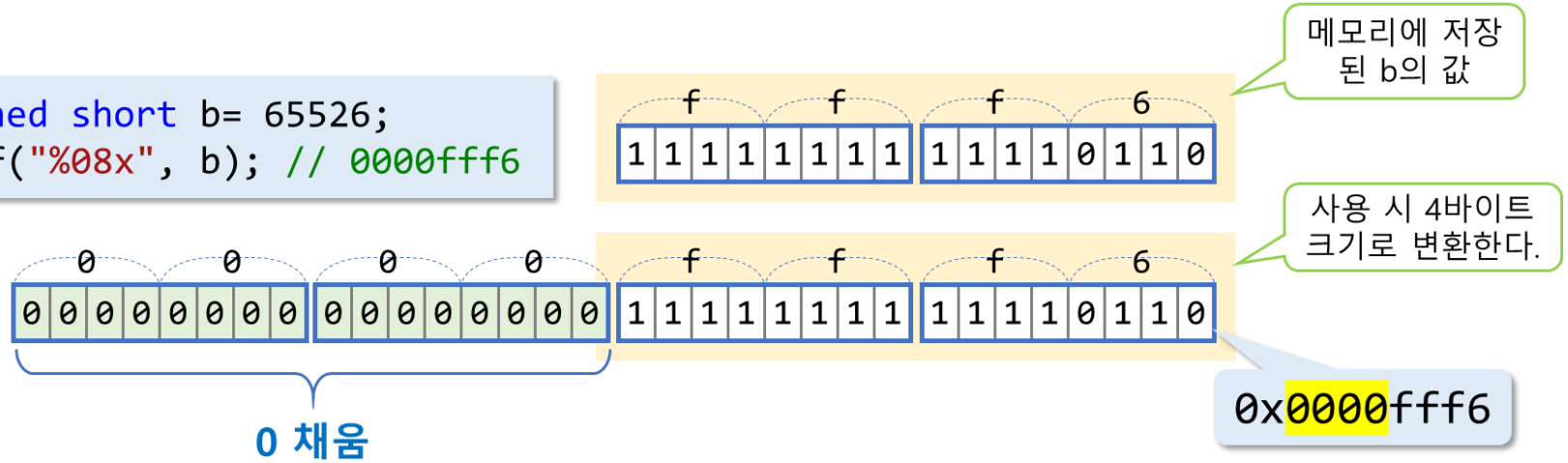
정수의 승격

- char, short형 변수는 사용 시 int형으로 변환된다

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




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



정수형으로 사용되는 char형

❖char형은 1바이트 크기의 정수형으로 사용

- char형의 범위: -128~127
  - 작은 크기의 정수를 저장할 때 유용
- char형도 정수형이므로 덧셈, 뺄셈과 같은 연산 가능

```
char n = 127;    // 정수형으로 사용되는 char형

printf("n   = %d\n", n);        // 정수처럼 10진수로 출력할 수 있다.
printf("n+1 = %d\n", n + 1);    // 정수처럼 덧셈 연산을 할 수 있다.
```

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

```
unsigned char control_flag = 0x47;     // 1바이트 크기의 컨트롤 플래그 0100 0111
unsigned char image[256];              // 256바이트 크기의 이미지
```

- char 형 변수는 문자와 정수 값을 저장하는 데 사용됨
- 보통 각 char는 메모리의 1 바이트에 저장됨
  - 256개의 값을 저장할 수 있음
  - 이 값들 중 일부분만이 실제 인쇄 문자임
    (소문자, 대문자, 숫자, 마침표, 특수문자 등)
  - 공백, 탭, 개행 문자 같은 여백도 있음
- 대부분의 컴퓨터는 ASCII나 EBCDIC 문자 코드 사용

예제 3-4 : char형과 unsigned char형의 오버플로우, 언더플로우

```c
03  int main(void)
04  {
05      char n = 128;           // n에 유효 범위 밖의 값을 저장한다.
06      unsigned char m = 256;  // m에 유효 범위 밖의 값을 저장한다.
07      char x = -129;          // x에 유효 범위 밖의 값을 저장한다.
08      unsigned char y = -1;   // y에 유효 범위 밖의 값을 저장한다.
09
10      printf("n = %d\n", n);
11      printf("m = %d\n", m);
12      printf("x = %d\n", x);
13      printf("y = %d\n", y);
14
15      return 0;
16  }
```

실행결과

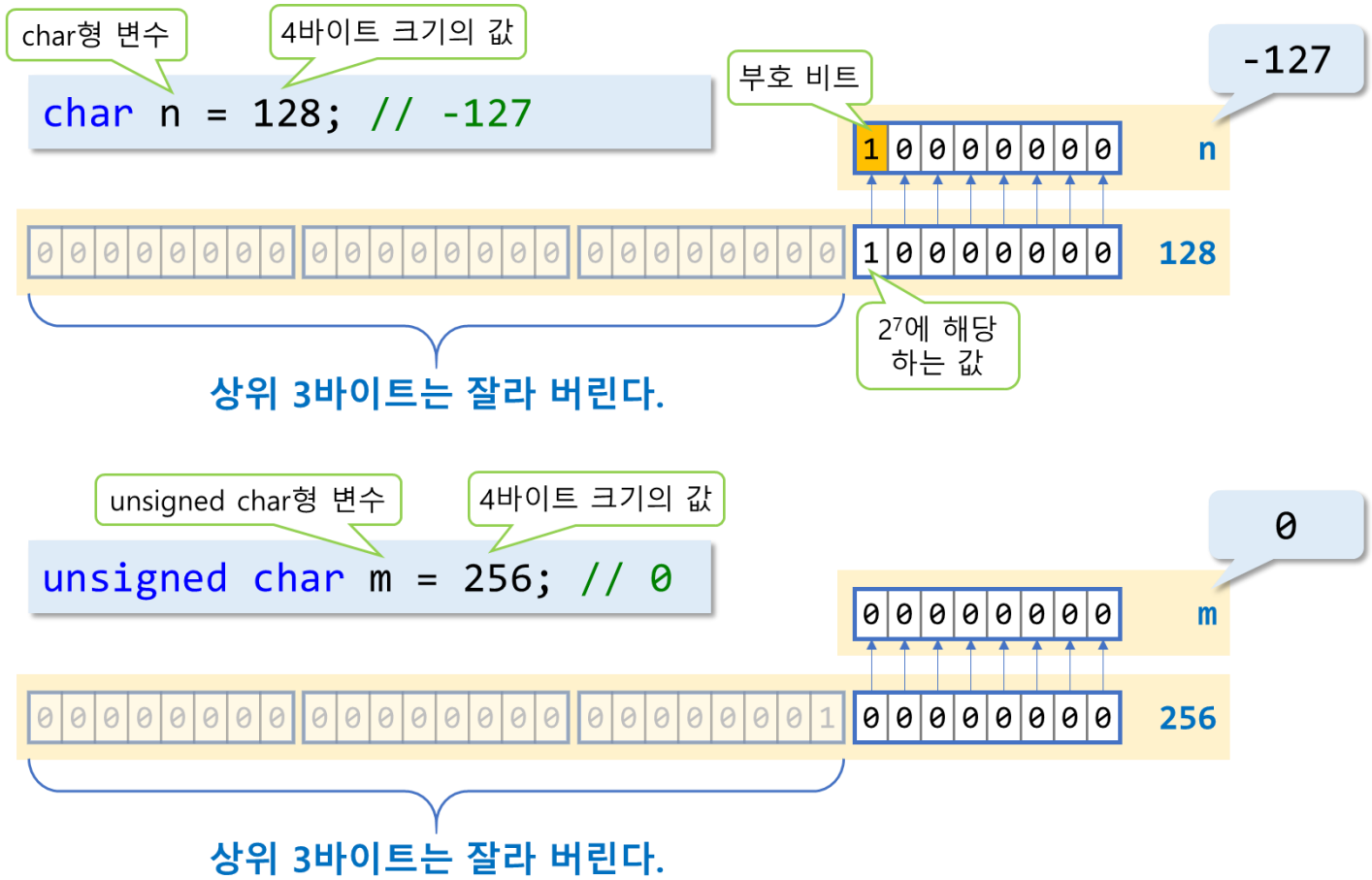
```
n = -128
m = 0
x = 127
y = 255
```

오버플로우

언더플로우

정수형의 유효 범위

- 정수형 변수에 유효 범위 밖의 값을 저장하면, 정수형의 크기에 맞춰 값의 나머지 부분을 잘라 버리고 유효 범위 내의 값만 저장된다



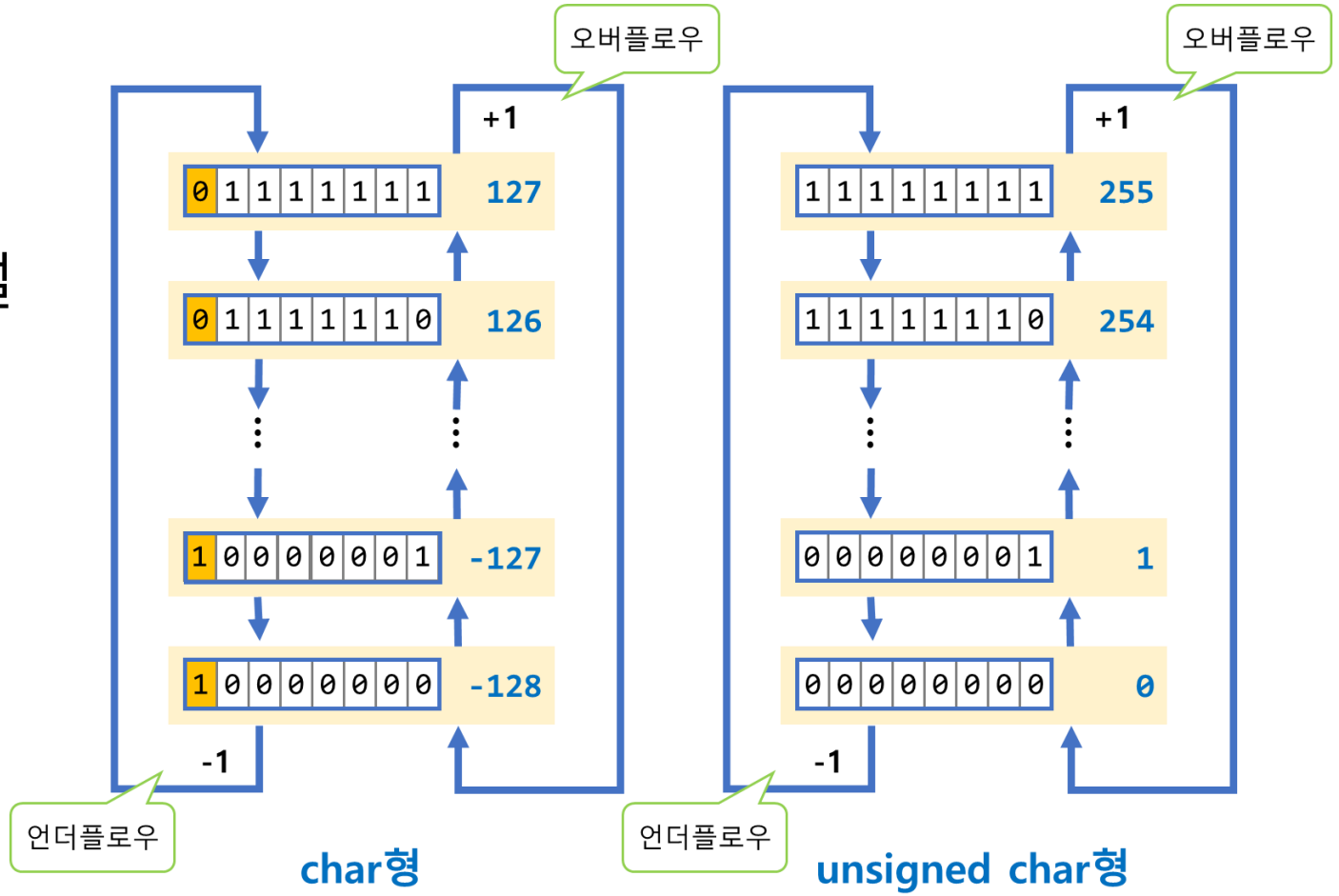
오버플로우와 언더플로우

❖ 오버플로우

- 정수형의 최대값보다 큰 값을 저장할 때 값이 넘쳐흘러서 유효 범위 내의 값으로 설정되는 것

❖ 언더플로우

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




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

문자형 (Char 자료형)

❖문자의 2진 표현 : ASCII 코드

- 제어 문자 : 0~31, 127
- 출력 가능한 문자 : 32~126

❖특수 문자

- ' '안에 역슬래시(\)와 함께 정해진 문자를 이용해서 나타낸다
- '\' 다음에 8진수로 적어주거나 '\x' 다음에 16진수로 적어준다

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)

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

- ASCII 코드에서 문자 상수와 대응되는 정수 값

문자 상수	'a'	'b'	'c'	…	'z'
대응하는 값	97	98	99	…	122
문자 상수	'A'	'B'	'C'	…	'Z'
대응하는 값	65	66	67	…	90
문자 상수	'0'	'1'	'2'	…	'9'
대응하는 값	48	49	50	…	57
문자 상수	'&'	'*'	'+'		
대응하는 값	38	42	43		

예제 3-5 : 입력된 문자의 ASCII 코드 출력

```
03  int main(void)
04  {
05      char ch, prev_ch, next_ch;
06
07      printf("문자? ");
08      scanf("%c", &ch);          // 문자 입력
09
10      prev_ch = ch - 1;          // ch 이전 문자
11      next_ch = ch + 1;          // ch 다음 문자
12      printf("prev_ch = %c, %d, %#02x\n", prev_ch, prev_ch, prev_ch);
13      printf("ch      = %c, %d, %#02x\n", ch, ch, ch);
14      printf("next_ch = %c, %d, %#02x\n", next_ch, next_ch, next_ch);
15
16      return 0;
17  }
```

ASCII 코드를 0x로 시작하는 16진수로 출력한다.

실행결과

```
문자? X
prev_ch = W, 87, 0x57
ch      = X, 88, 0x58
next_ch = Y, 89, 0x59
```

예제 3-6 : 특수 문자

```c
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.
```

정수형 (int 자료형)

- C 언어의 기본적인 자료형
- int 형은 2 바이트(= 16 비트)나 4 바이트(= 32 비트)에 저장됨
  - 즉, 정수의 일부분만 저장할 수 있음

<u>값의 범위</u>

❖4 바이트 워드 컴퓨터

- 최소 = -2^{31} = -2147483648
- 최대 = $+2^{31} - 1$ = +2147483647

❖2 바이트 워드 컴퓨터

- 최소 = -2^{15} = -32768
- 최대 = $+2^{15} - 1$ = +32767

❖정수 오버플로

- 값의 범위를 초과할 때 발생 (주의 필요)

```
#define BIG 2000000000     /* 2 billion */

int main(void){
   int a, b = BIG, c = BIG;

   a = b + c;         /* out of range? */

   .....
```

- 정수 오버플로가 발생해도 프로그램은 계속 수행되지만, 논리적으로 부정확한 값이 계산됨

정수형 short, long, unsigned

- short 자료형은 기억장소를 절약하고자 하는 경우에 사용
- long 형은 큰 정수 값을 다룰 때 사용
- short 형은 2 바이트에 저장되고, long 형은 4 바이트에 저장됨
  - 4 바이트 워드 컴퓨터: int 형과 long 형 크기 같음
  - 2 바이트 워드 컴퓨터: int 형과 short 형 크기 같음
- unsigned 형의 변수는 음수가 아닌 정수를 표현할 때 사용함

실수형 (부동형: float, double 자료형)

- 실수 값을 다루기 위해 사용함
- 정수를 부동형 상수로 표현할 때에는 소수점을 사용해야 함
  - 상수 1.0과 2.0은 double형이지만,
  - 상수 3은 int 형임
- 접미사

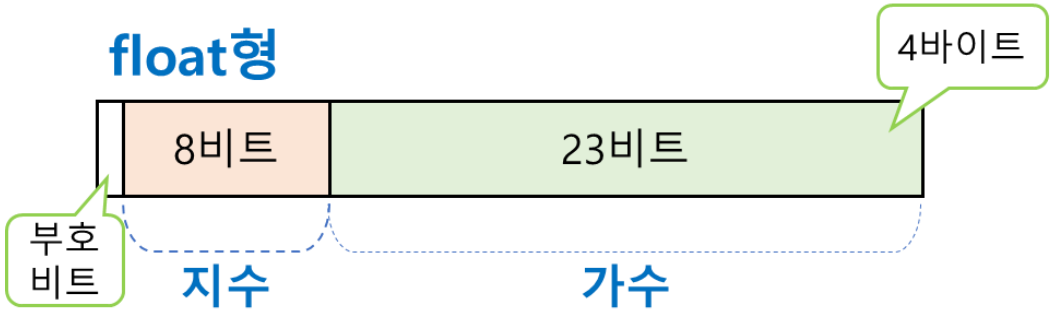
f 또는 F	**float**	**3.7F**
l 또는 L	**long double**	**3.7L**

- 부동소수점 방식

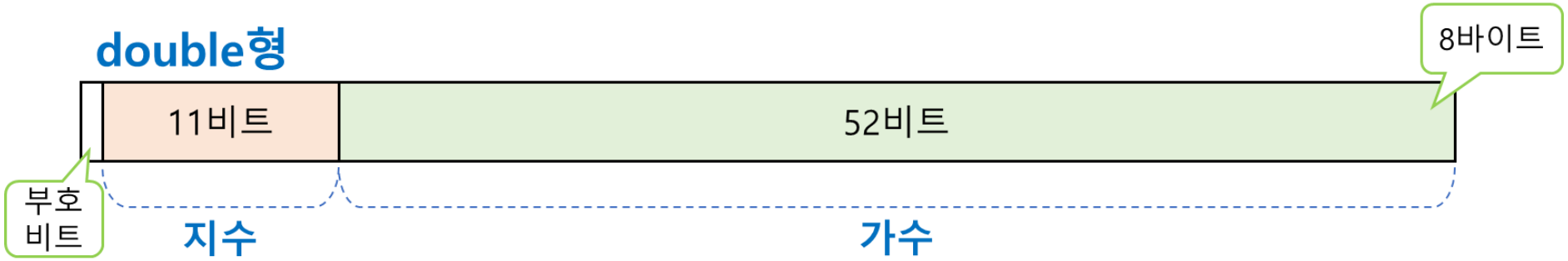
실수의 정밀도 — 가수 — 지수 — 실수의 범위

$$1.xxx \times 2^n$$

단정도



배정도



- 실수의 표현

0 1 2 . . . 8 9 . . . 31

부호	지수부	가수부(1 hidden bit)

- 예

$2.0 = 1.0 \times 2^1$: 01000000 00000000 00000000 00000000

$4.0 = 1.0 \times 2^2$: 01000000 10000000 00000000 00000000

$-2.0 = -1.0 \times 2^1$: 11000000 00000000 00000000 00000000

값의 범위

❖부동형에 배정될 수 있는 값은 *정밀도와 범위라는* 속성으로 기술됨

- 정밀도 : 부동형 값이 가질 수 있는 유효숫자 수
- 범위 : 부동형의 변수에 저장될 수 있는 가장 큰 양수와 가장 작은 양수의 한계

❖float 형

- 정밀도 : 대략 유효숫자 6자리
- 범위 : 대략 10^{-38}에서 10^{+38}

❖double 형

- 정밀도 : 대략 유효숫자 15자리
- 범위 : 대략 10^{-308}에서 10^{+308}

예제 3-7 : 실수형의 정밀도

```
03  int main(void)
04  {
05      float  pi1 = 3.141592653589793;
06      double pi2 = 3.141592653589793;
07
08      printf("float  형의 pi 값 : %30.25f\n", pi1);
09      printf("double 형의 pi 값 : %30.25f\n", pi2);
10
11      return 0;
12  }
```

소수점 이하 15자리인 실수 값

유효숫자 30개중 소수점 이하 25자리 출력

실행결과

```
float  형의 pi 값 :    3.1415927410125732421875000
double 형의 pi 값 :    3.1415926535897931159979635
```

float형은 소수점 이하 6자리까지만 올바르게 출력된다.

double형은 소수점 이하 15자리까지 올바르게 출력된다.

실수의 유효 범위

❖실수형의 오버플로우

- 최대값보다 큰 값을 저장하려고 하면 무한대를 의미하는 INF로 설정

❖실수형의 언더플로우

- 최소값보다 작은 값을 저장하려고 하면, 가수 부분을 줄이고 지수 부분을 늘려서 실수를 표현하거나, 만일 그것이 불가능해지면 0으로 만들어 버린다

❖실수형의 크기와 유효 범위

데이터형		크기	유효 범위
실수형	float	4바이트	$\pm 1.17549 \times 10^{-38} \sim \pm 3.40282 \times 10^{38}$
	double	8바이트	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{308}$
	long double	8바이트	$\pm 2.22507 \times 10^{-308} \sim \pm 1.79769 \times 10^{308}$

예제 3-8 : float형의 오버플로우와 언더플로우

```
03  int main(void)
04  {
05      float x = 3.40282e38;     // float형의 최대값
06      float y = 1.17549e-38;    // float형의 최소값
07
08      printf("x = %30.25e\n", x);
09      printf("y = %30.25e\n", y);
10
11      x = x * 100;    // float형의 오버플로우
12      printf("x = %30.25e\n", x);
13
14      y = y / 1000;   // 가수부를 줄여서 실수 표현
15      printf("y = %30.25e\n", y);
16
17      y = y / 1000;   // 가수부를 줄여서 실수 표현
18      printf("y = %30.25e\n", y);
19
20      y = y / 1000;   // float형의 언더플로우
21      printf("y = %30.25e\n", y);
22
23      return 0;
24  }
```

지수표기로 출력

실행결과

```
x = 3.4028200183756559773330698e+38
y = 1.1754900067970481010358167e-38
x =                            inf
y = 1.1755492817220890407979167e-41
y = 1.1210387714598536567389837e-44
y = 0.0000000000000000000000000e+00
```

변수의 필요성

- 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.

변수의 선언과 사용

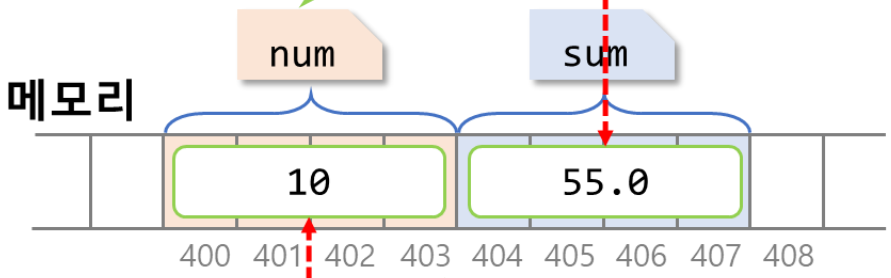
변수가 사용할 메모리 할당

입력받은 값을 변수에 저장

연산의 결과를 변수에 저장

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

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

  return 0;
}
```

변수의 이름을 사용해서 접근



변수의 주소는 알 필요 없다.

[실행 결과]

```
10
sum = 55.0
```

변수의 선언

❖변수를 선언하려면 변수의 데이터형과 이름이 필요하다

형식

데이터형 변수명;
데이터형 변수명1, 변수명2, … ;

사용예

```
float discount_rate;
int width, height;
unsigned char red, green, blue;
```

❖식별자를 만드는 규칙

- 반드시 영문자, 숫자, 밑줄 기호(_)만을 사용해야 한다
- 첫 글자는 반드시 영문자 또는 밑줄 기호(_)로 시작해야 한다. 식별자는 숫자로 시작해서는 안 된다
- 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

변수의 선언 예

- 올바른 변수 선언의 예

```
int salary2018;      // 변수명의 첫 글자 외에는 숫자를 사용할 수 있다.
double _rate;        // 변수명은 _로 시작할 수 있다.
int width, height;   // 여러 개의 변수를 선언할 때는 ,를 사용한다.
long discount_rate;  // 여러 단어를 연결할 때는 _를 사용한다.
int discountRate;    // 연결되는 단어의 첫 글자를 대문자로 지정한다.
```

- 잘못된 변수 선언의 예

```
long text-color;     // 변수명에 - 기호를 사용할 수 없다.
int elapsed time;    // 변수명에 빈칸을 포함할 수 없다.
int 2018salary;      // 변수명은 숫자로 시작할 수 없다.
char case;           // C 키워드는 변수명으로 사용할 수 없다.
```

변수 선언문의 위치

ANSI C

```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.

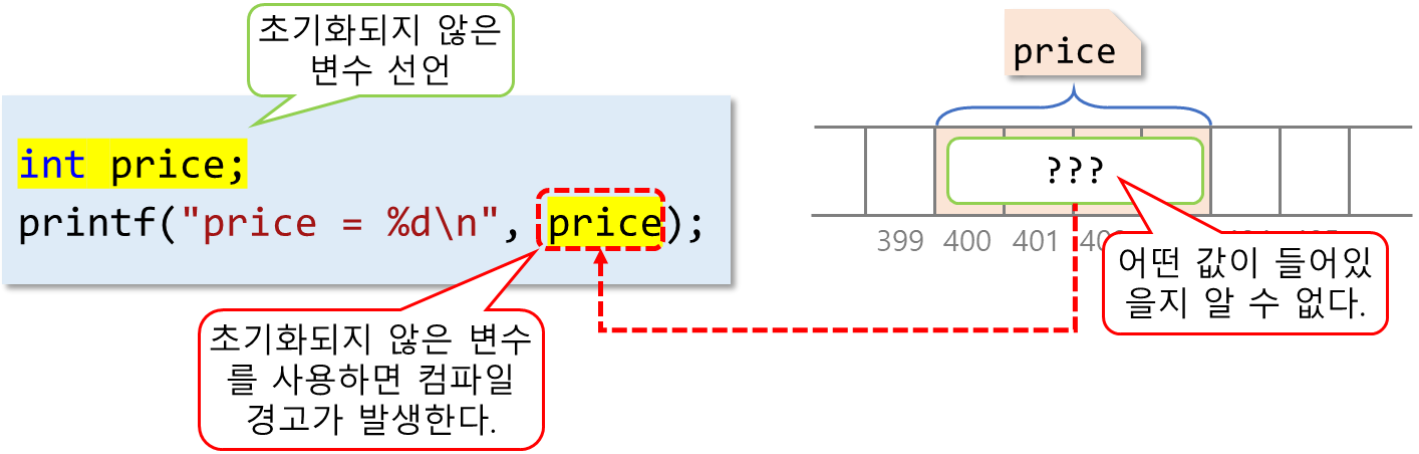
C99

```c
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.

변수의 초기화 (1/2)

- 선언 시 초기화되지 않은 변수의 값은 쓰레기 값이다



- 변수의 초기화 방법

형식

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

사용예

```
float discount_rate = 0.1;
char size = 'L';
int width = 100, height = 100;
```

변수의 초기화 (2/2)

- 변수를 초기화할 때 변수의 데이터형과 같은 형의 값으로 초기화해야 한다

```
int area = 3.14 * 5 * 5;        // 정수형 변수를 실수 값으로 초기화하므로 컴파일 경고 발생
float discount_rate = 0.1;      // 0.1은 double형이므로 컴파일 경고 발생
```

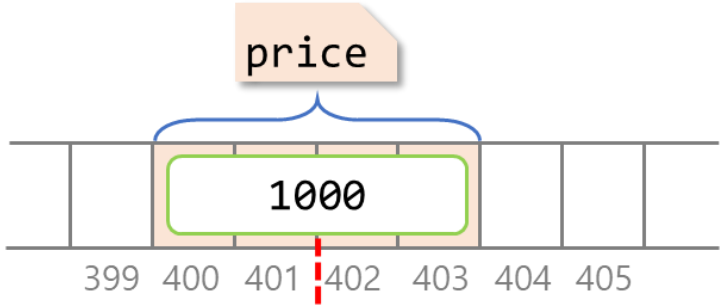
- 변수가 어떤 초기값을 가져야 하는지 알 수 없을 때에는 0으로라도 초기화하는 것이 안전하다

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

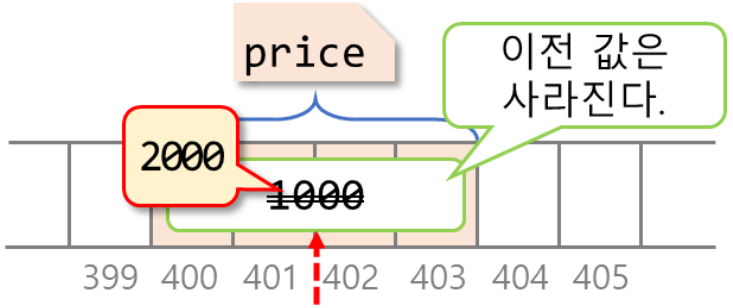
변수의 사용

```
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 ;
```



예제 3-9 : 변수의 선언 및 사용

```
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

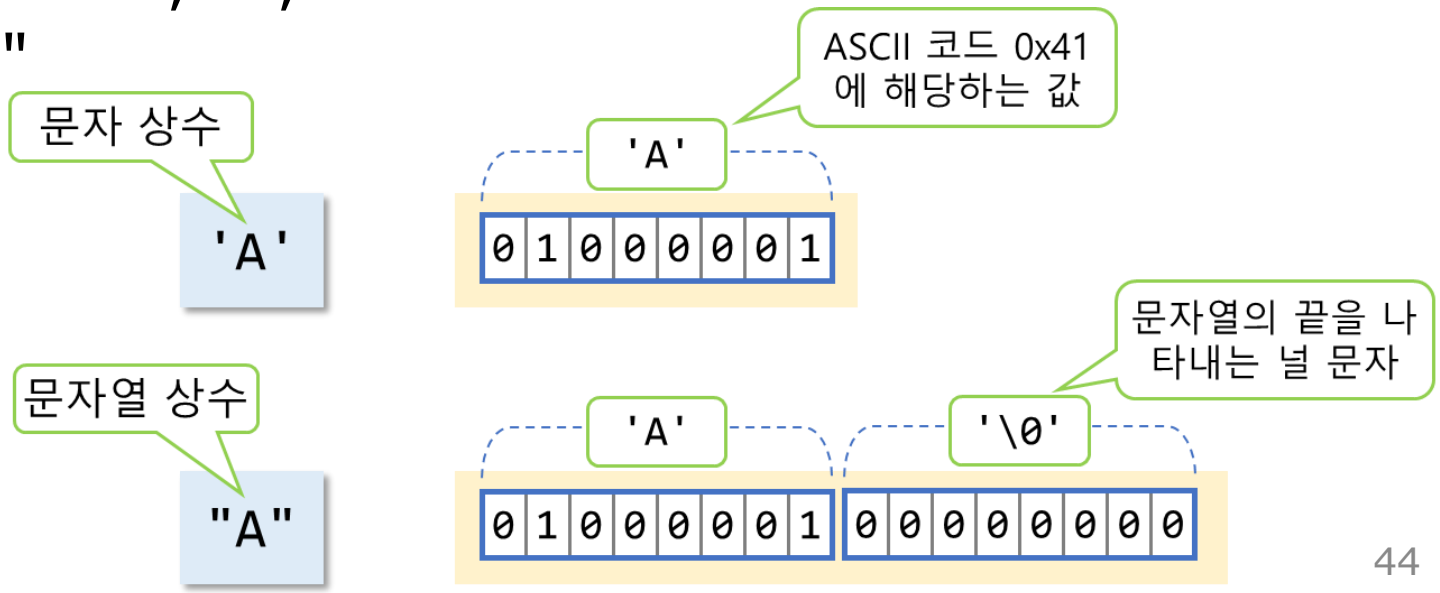
```
amount = -858993460, price = 0
수량? 2
합계: 4000원
```

쓰레기 값

상수

❖프로그램에서 값이 변경되지 않는 요소
- 메모리에 저장되지 않고, 한 번만 사용된 다음 없어져 버리는 임시 값

❖리터럴 상수 : 값 자체
- 문자 상수 : 'A', '\xa'
- 정수형 상수 : 0x12, 123u, 1234567L
- 실수형 상수 : 12.34, 1.234e1, .1, 0.5F
- 문자열 상수 : "hello", "A"



예제 3-10 : 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

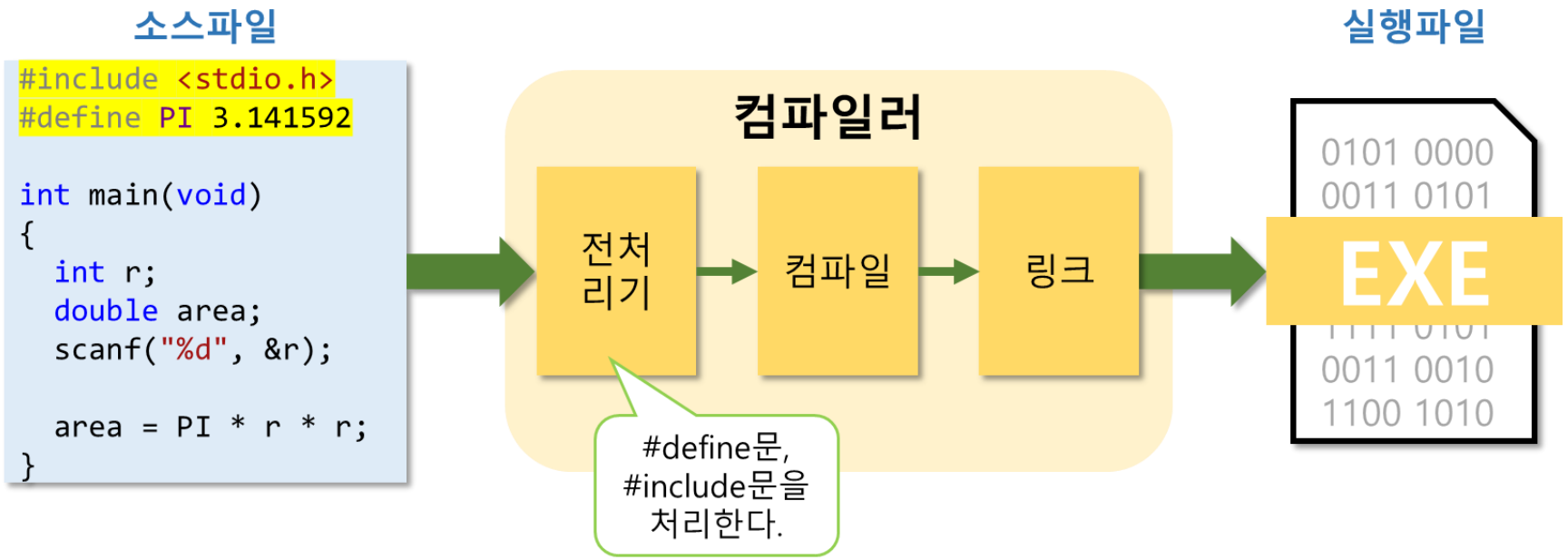
실행결과

```
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
```

매크로 상수

❖ #define문으로 정의되는 상수

- 전처리기가 처리하는 문장
- 전처리기는 컴파일러가 소스 파일을 컴파일하기 전에 먼저 수행되며, 프로그래머가 작성한 소스 파일을 컴파일할 수 있도록 변환해서 준비한다



전처리기의 #define 처리

❖전처리기는 매크로 상수를 특정 값으로 대치(replace)한다

- 문자열 상수 안에 포함된 매크로 상수는 문자열의 일부이므로 대치되지 않는다

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.

예제 3-11 : 매크로 상수

```
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
09      printf("반지름? ");
10      scanf("%lf", &radius);  // double형 입력
11
12      area = PI * radius * radius;
13      printf("원의 면적: %.2f\n", area);
14      printf("PI = %.2f\n", PI);
15      return 0;
16  }
```

3.14로 대치

3.14로 대치

문자열이므로 대치되지 않는다.

실행결과

```
반지름? 5
원의 면적: 78.50
PI = 3.14
```

표준 C 라이브러리의 매크로 상수

limits.h

```
#define SCHAR_MIN    (-128)
#define SCHAR_MAX      127
#define UCHAR_MAX      0xff
#define CHAR_MIN     SCHAR_MIN
#define CHAR_MAX     SCHAR_MAX
#define SHRT_MIN     (-32768)
#define SHRT_MAX       32767
#define USHRT_MAX      0xffff
#define INT_MIN      (-2147483647 - 1)
#define INT_MAX        2147483647
#define UINT_MAX       0xffffffff
#define LONG_MIN     (-2147483647L - 1)
#define LONG_MAX       2147483647L
#define ULONG_MAX      0xffffffffUL
#define LLONG_MAX      9223372036854775807i64
#define LLONG_MIN    (-9223372036854775807i64 - 1)
#define ULLONG_MAX     0xffffffffffffffffui64
```

예제 3-12 : <limits.h>와 <float.h>를 이용한 오버플로우 발생

```c
#include <stdio.h>
#include <limits.h> // 정수형의 최대 최소를 나타내는 매크로 상수가 정의된 헤더 파일
#include <float.h>  // 실수형의 최대 최소를 나타내는 매크로 상수가 정의된 헤더 파일

int main(void)
{
    char a = CHAR_MAX;
    char b = CHAR_MAX + 1;   // 오버플로우
    short c = SHRT_MAX;
    short d = SHRT_MAX + 1;  // 오버플로우
    int e = INT_MAX;
    int f = INT_MAX + 1;     // 오버플로우
    float g = FLT_MAX;
    float h = FLT_MAX * 10;  // 오버플로우

    printf("a = %d, b = %d\n", a, b);
    printf("c = %d, d = %d\n", c, d);
    printf("e = %d, f = %d\n", e, f);
    printf("g = %30.25e, h = %30.25e\n", g, h);

    return 0;
}
```

매크로 상수가 정의된 라이브러리 헤더 포함

실행결과

```
a = 127, b = -128
c = 32767, d = -32768
e = 2147483647, f = -2147483648
g = 3.4028234663852885981170418e+38, h =                    inf
```

const 변수

- 값을 변경할 수 없는 변수

```
const int MAX_BUF = 32;
MAX_BUF = 128;      // const 변수는 변경할 수 없으므로 컴파일 에러 발생
```

- const 변수는 반드시 선언 시 초기화해야 한다

```
const float VAT_RATE;          // 초기화 안하면 컴파일 에러는 아니지만 사용할 수 없게 된다.
VAT_RATE = 0.1;                // const 변수에 대입을 할 수 없으므로 컴파일 에러 발생
```

예제 3-13 : const 변수의 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
```

기호 상수를 사용해야 하는 이유 (1/2)

- 기호 상수를 사용하면 프로그램을 수정하기 쉽다

리터럴 상수를 사용하는 경우

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

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

3.141592

3.141592

리터럴 상수를 사용하는
모든 곳을 수정해야 한다.

기호 상수를 사용하는 경우

```
#define PI 3.14

double area, perimeter;
int r = 5;
area = PI * r * r;
perimeter = 2 * PI * r;
```

3.141592

기호 상수를 정의하는 곳만
수정하면 된다.

기호 상수를 사용해야 하는 이유 (2/2)

- 기호 상수를 사용하면 프로그램이 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.

형 변환

- 일시적으로 변수의 자료형을 다른 타입으로 변경하는 것
- *(dataType)value*

- (예) (float)age;
- (예) salaryBonus = salary * (float)age / 150.0;
- (예) value = (float)(number - 10 * yrsService);

캐스트

- 명시적인 변환

- 캐스트 예제

```
(double) i
(long) ('A' + 1.0)
x = (float) ((int) y + 1)
(double) (x = 77)
```

- 잘못된 캐스트

```
(double) x = 77
/* equivalent to ((double) x) = 77, error */
```

형 변환 연산자

- 명시적인 형 변환

형식	(데이터형) 수식

사용예	`(double) 0`
	`(int) price * rate`
	`(double) 10 / (double) 20`

```
int result1, result2;
result1 = (int)(1.5 + 3.8);          // 1.5 + 3.8을 먼저 수행한 다음 (int) 5.3
result2 = (int)1.5 + (int)3.8;       // (int)1.5와 (int)3.8을 먼저 수행한 다음 1 + 3
```

질의 및 응답

참고문헌

- 천정아, 『Core C Programming』, 연두에디션(2019)
- C가 보이는 그림책, ANK Co., Ltd. , 성안당 (2018)
- Greg Perry, Dean Miller 『어서와 C언어는 처음이지』, 천인국 옮김, 인피니티북스(2015)
- KELLEY (역 : 김명호 외), 『A Book on C』, 홍릉과학출판사 (2003)
- 윤성우, 『 열혈 C 프로그래밍』, 오렌지미디어
- 천인국, 『 쉽게 풀어쓴 C언어 Express』, 생능출판사
- 서현우, 『 뇌를 자극하는 C 프로그래밍』, 한빛미디어
- 강성수, 『 쾌도난마 C프로그래밍』, 북스홀릭
- 고응남, 『 C프로그래밍 기초와 응용실습』, 정익사