<2장> 자료형, 변수, 상수, printf() 함수

학습 목표

- 자료형의 개념을 파악한다.
- C 언어에서 제공하는 기본 자료형을 알아본다.
- 변수의 개념과 용도를 파악한다.
- 대입문, 변수의 초기화 등을 학습한다.
- 화면에 값을 출력하는 printf() 함수의 사용법을 익힌다.
- typedef를 이용해 기존 자료형에 새로운 이름을 붙이는 방법을 알아본다.
- 상수와 열거형의 사용법을 살펴본다.

목차

01

자료형

1. 자료형

자료형의 필요성

- 메모리에 있는 값을 사용하기
  - 메모리에 있는 값이 정수, 소수점이 있는 값 또는 문자인지 어떻게 구별할 수 있을까?
  - 값이 정수라는 것을 알더라도, 메모리 어디부터 어디까지 있는 값인지 어떻게 알까?

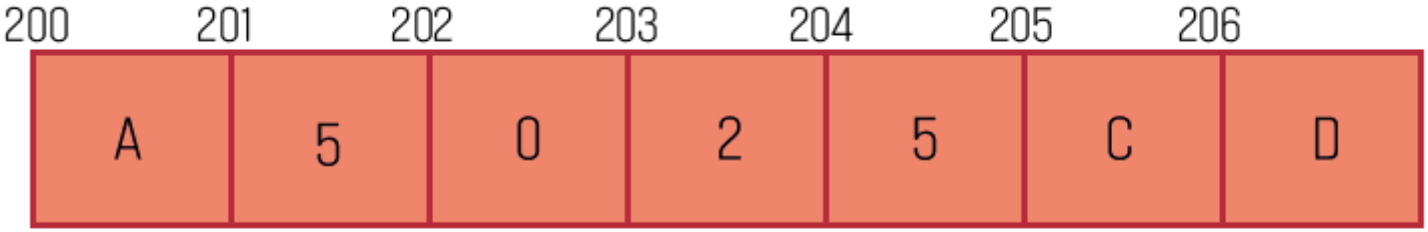

그림 2-1 메모리 값에 저장되어 있는 문자

1. 자료형

자료형의 의미와 목적

- 자료형은 값의 종류, 메모리 저장 공간 크기, 데이터의 표현 범위 결정
- C 언어는 기본적으로 정수, 실수, 문자 등을 구별할 수 있도록 자료형 정의
- 그 밖에 새로운 자료형을 만들어 내는 사용자 정의 자료형(user-defined data type)도 존재

1. 자료형

컴퓨터에서 문자 처리

- 사용자 → 컴퓨터로 입력
  - 컴퓨터는 이진수만 이해하므로 문자는 알 수 없고 이진수로 변환

- 컴퓨터 → 사용자로 출력
  - 메모리나 다른 저장 매체에 있는 이진수 값은 문자 형태로 변환

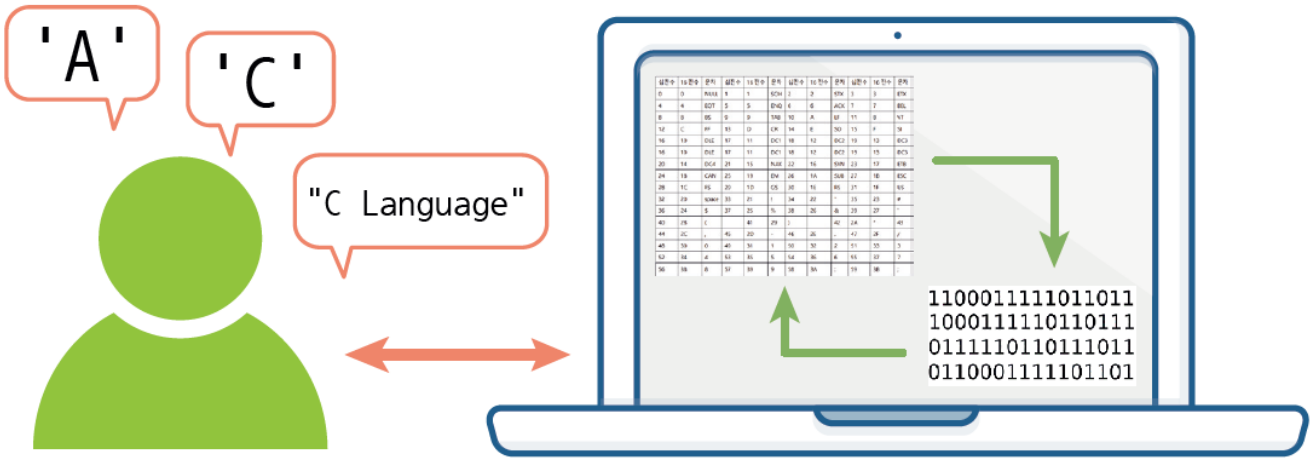

그림 2-2 사용자와 컴퓨터의 입력/출력

1. 자료형

■ 컴퓨터에서 문자 처리

▪ 아스키 문자표

- 1960년대 미국에서 만들어짐
- 초기 코드표는 화면에 보이지 않는 제어 문자, 영문 대/소문자, 0~9까지의 숫자, 특수 문자들을 포함해서 128글자(7 bit)를 표현

표 2-1 아스키 문자표의 일부

십진수	이진수	문자	십진수	이진수	문자	십진수	이진수	문자
60	00111100	〈	61	00111101	=	62	00111110	〉
63	00111111	?	64	01000000	@	65	01000001	A
66	01000010	B	67	01000011	C	68	01000100	D
69	01000101	E	70	01000110	F	71	01000111	G

1. 자료형

상수의 개념과 표현 방법

- 상수(constant)란 코드가 실행되면서 변경되지 않는 값

리터럴 상수(literal constant)

- 리터럴(literal)이라고도 하며, 정수, 실수, 문자, 문자열 상수가 포함
- 코드에 직접 작성하는 숫자, 문자, 문자열 등이 리터럴 상수

정수 상수 표현 방법

- C 코드에서 정수는 10진수, 16진수, 8진수, 2진수로 표현 가능
  - 10진수(decimal number) : 수학의 정수 표현 방법과 동일
  - 16진수(hexadecimal number) : 정수 앞에 '0x' 또는 '0X'를 따옴표 없이 붙임
  - 8진수(octal number) : 정수 앞에 숫자 0을 붙임
  - 2진수(binary number) : 정수 앞에 '0b' 또는 '0B'를 붙임

1. 자료형

상수의 개념과 표현 방법

- 실수 상수 표현 방법
  - 실수 - 소수점이 있는 숫자
  - C 코드 - 소수점 앞뒤에 있는 0 생략 가능
  - 십진수로 표기하기에 너무 긴 실수는 숫자와 10의 거듭제곱의 형식으로 표현 가능

표 2-2 2243.3을 다양한 형태의 C 코드로 표현

지수 형태의 실수	C 코드에서의 표현 방법
224.33×10^{1}	224.33E1 또는 224.33e1
22.433×10^{2}	22.433E2 또는 22.433e2
2.2433×10^{3}	2.2433E3 또는 2.2433E3
0.22433×10^{4}	0.22433E4, 0.22433e4, .22433E4 또는 .22433e4
22433.00×10^{-1}	22433.00E-1 또는 22433.00e-1

1. 자료형

■ 상수의 개념과 표현 방법

▪ 문자 상수 표현 방법

- 문자 한 개를 표현하는 방법
- 작은따옴표(') 사이에 문자를 표시
  - 예시 : a(소문자 a), A(대문자 A), "(큰따옴표), ,(콤마), ' '(공백 문자)

▪ 문자열 상수 표현 방법

- 큰따옴표(") 사이에 0개 이상의 글자들 표시
- 큰따옴표 사이에 글자가 없어도 문자열(빈 문자열)

코드 2-1 문자열 상수의 예

```
1   ""                  // 빈 문자열도 문자열 상수
2   "a"                 // 문자가 1개만 있어도 큰따옴표 사이에 있으면 문자열
3   "hello" " world"    // "hello world"와 동일한 문자열
4   "'a' as in apple"   // 큰따옴표 안에 쓰는 작은따옴표
5   "hello"
6   "world"             // 두 줄에 써도 "hello world"와 동일한 문자열 상수
```

1. 자료형

■ 상수의 개념과 표현 방법

- 그림 2-3처럼 문자열 안에 따옴표를 표현해야 한다면 어떻게 할까?

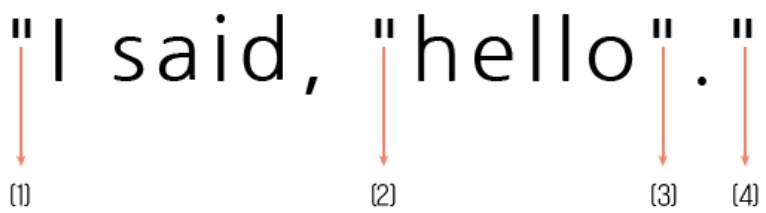

그림 2-3 이스케이프 시퀀스

- 이스케이프 시퀀스
  - 필요한 글자들의 표현을 위해 역슬래시(₩)와 문자를 조합해 구성한 글자

표 2-4 주요 이스케이프 시퀀스 문자

이스케이프 시퀀스	설명
\0	널(null) 문자. 아스키 문자표의 0번 문자. C 언어에서는 문자열의 끝을 나타내기 위해 사용
\\	역슬래시(backslash) 문자(\)
\'	작은따옴표(')
\"	큰따옴표(")
\n	줄바꿈 문자. 개행문자라고도 부름. 줄을 바꿔서 출력하는 목적으로 사용
\t	탭 문자. 아스키 문자표의 9번(Tab) 문자. 일정 공간을 띄어쓰기를 하기 위한 목적으로 사용

상수의 개념과 표현 방법

이스케이프 시퀀스

- 이스케이프 시퀀스의 사용 예시

코드 2-2 이스케이프 시퀀스를 사용한 문자 또는 문자열 상수

```
1  '\n'          // 줄바꿈 문자
2  '\\'          // 역슬래시 문자
3  '"'           // 문자 상수로 표현하는 큰따옴표는 이스케이프 시퀀스를 사용하지 않아도 됨
4  '\"'          // 문자 상수로 표현하는 큰따옴표는 이스케이프 시퀀스를 사용해도 문제는 없음
5  '\''          // 작은따옴표 문자
6  "\\"          // 역슬래시 문자열
7  "'a' as in apple" // 문자열 상수에서 '는 이스케이프 시퀀스를 사용하지 않아도 됨
8  "\'a\' as in apple" // 문자열 상수에서 '를 이스케이프 시퀀스로 표현하는 것도 괜찮음
9  "hello\tworld\n" // hello와 world 사이를 tab 문자로 띄우고, 마지막에 줄바꿈 문자를 붙임
10 "I said, \"hello\"." // I said, "hello". 문자열에 큰따옴표가 포함되면 이스케이프 시퀀스 사용
```

1. 자료형

■ 기본 자료형

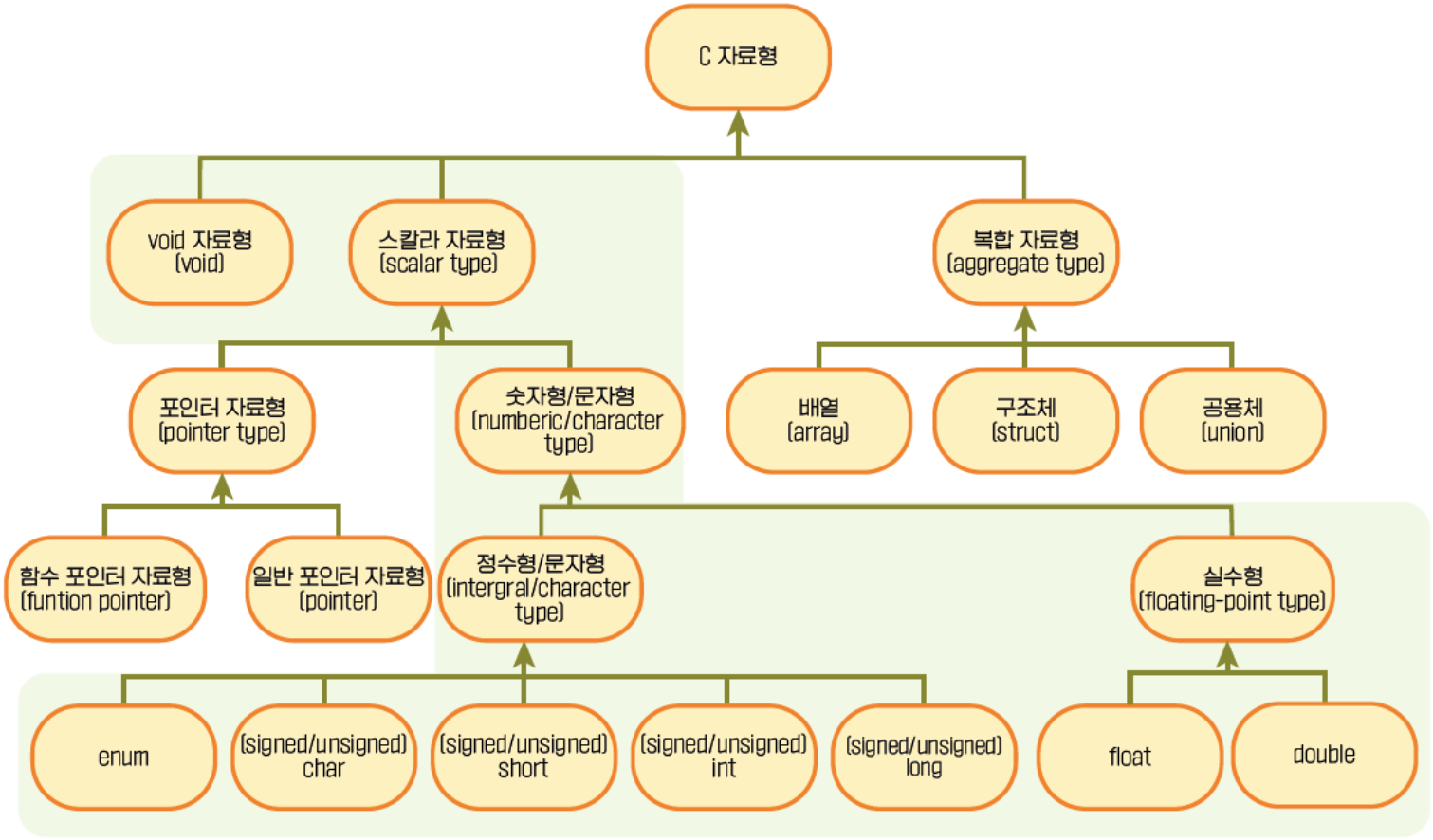

그림 2-4 C 언어의 자료형

1. 자료형

기본 자료형

- void 자료형(나중에 학습함. 지금은 자세히 몰라도 됨)
  - 특수한 경우에 사용하는 자료형

표 2-5 void 자료형을 사용하는 경우

사용하는 경우	설명	예제 코드
함수의 인자로 사용	인자로 전달되는 값이 없음	`int main(void) {}`
함수의 반환 자료형으로 사용	함수가 값을 반환하지 않음	`void f() {}`
포인터로 사용	어떤 자료형의 주소값도 저장 가능	pn이 정수형 포인터 변수라고 가정할 때 `void* p = pn;`

- 스칼라 자료형
  - 값을 한 개만 가질 수 있고 복합(aggregate) 자료형은 값을 여러 개 저장 가능

1. 자료형

기본 자료형

정수 자료형

- 정수(0, 양수, 음수)
- 불린(boolean) 값 – 참(true) 또는 거짓(false)을 나타내는 값
- 문자
- 열거형(enum)

1. 자료형

기본 자료형

정수 자료형

- 정수
  - 정숫값을 표현할 수 있는 범위, 메모리 사용량에 따라 다양한 정수 자료형 제공
  - unsigned 또는 signed 수식어에 따라 다르게 취급

표 2-5 정수형이 사용하는 메모리 공간의 크기와 표현할 수 있는 값의 범위

자료형	메모리 공간 크기	표현 범위
signed int	4바이트	$-2147483648 \sim 2147483647$ $(-2^{31} \sim 2^{31} - 1)$
unsigned int	4바이트	$0 \sim 4294967295$ $(0 \sim 2^{32} - 1)$
signed long	4바이트	$-2147483648 \sim 2147483647$ $(-2^{31} \sim 2^{31} - 1)$
unsigned long	4바이트	$0 \sim 4294967295$ $(0 \sim 2^{32} - 1)$
signed short	2바이트	$-32768 \sim 32767$ $(-2^{15} \sim 2^{15} - 1)$
unsigned short	2바이트	$0 \sim 65535$ $(0 \sim 2^{16} - 1)$
signed long long	8바이트	$-9223372036854775808 \sim 9223372036854775807$ $(-2^{63} \sim 2^{63} - 1)$
unsigned long long	8바이트	$0 \sim 18446744073709551616$ $(0 \sim 2^{64} - 1)$

1. 자료형

기본 자료형

정수 자료형

- int 자료형
  - 정수 자료형(integral type)에서 가장 기본이 되는 자료형
  - 컴퓨터에서 가장 빨리 처리하고 연산할 수 있는 값에 해당되는 메모리 크기를 할당
- short int와 long int 자료형
  - int 형에서 파생된 자료형
  - short와 long으로 주로 사용
- long long int 자료형
  - long long int도 주로 long long으로 사용
- 정수 상수와 자료형
  - 코드에서 int 형 범위 안에 있는 값을 정수 상수로 사용하면, 컴파일러가 int 자료형으로 취급

기본 자료형

정수 자료형

- sizeof 연산자

코드 2-3 Sizeof1.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       printf("%zu\n", sizeof(int));
6       printf("%zu\n", sizeof(long long));
7       printf("%zu\n", sizeof(1));
8       printf("%zu\n", sizeof(2.4));
9       return 0;
10  }
```

각각 int, long long, int 형 정수 상수 1, double 형 실수 2.4가 메모리에서 차지하는 공간의 크기를 바이트 단위로 화면에 출력

<실행 결과>

```
4
8
4
8
```

기본 자료형

정수 자료형

정수 상수와 자료형

표 2-6 정수 자료형을 명시하기 위해 상수 뒤에 붙이는 접미사

자료형	접미사	예
unsigned int	u 또는 U	100u 또는 100U
signed long	l 또는 L	100l 또는 100L
unsigned long	ul 또는 UL	100ul, 100uL, 100Ul, 100UL
long long	ll, LL	100ll 또는 100LL
unsigned long long	ull 또는 ULL	100ull 또는 100ULL

※ 상수가 차지하는 메모리 공간 크기 출력 코드

➔ 코드 2-4
➔ 실행 결과

1. 자료형

■ 기본 자료형

▪ 정수 자료형

- 불린 값 표현
- <stdbool.h>나 _Bool, bool 등은 C99

코드 2-5 Boolean.c

```
1   #include <stdbool.h> // bool, true, false를 사용하려면 필요함
2
3   int main(void)
4   {
5       _Bool b1 = 2 > 3; // 거짓 0 또는 false
6       bool b2 = 2 > 3;  // 거짓 false 또는 0
7       b2 = 1;           // 참 b2 = 1
8       b1 = false;       // 거짓 b1 = 0
9       int n1 = 3;       // 참 n1은 0이 아닌 다른 정수. n1
10      b2 = n1;
11      b1 = n1;
12  }
```

_Bool 형 b1에 0 저장

bool 형 b2에 0 저장

b2에 1 저장

b1에 0을 저장

b2와 b1에 정수 3을 저장

1. 자료형

기본 자료형

정수 자료형

- char 자료형(문자형)
  - char 자료형은 문자를 표현하는 자료형
  - 가장 작은 범위를 나타내는 정수 자료형이기도 함

```
'?'    'A'    'a'    '\t'    '\n'    '2'    '0'
```

기본 자료형

- 정수를 메모리에 저장하는 방법
  - signed 정수는 가장 상위 비트(MSB)를 부호 비트로 사용

그림 2-5 signed 정수

기본 자료형

- 정수를 메모리에 저장하는 방법
  - unsigned 정수는 전체 공간을 값을 저장하는 데 사용

그림 2-6 unsigned 정수

1. 자료형

기본 자료형

- 정수를 메모리에 저장하는 방법
  - size_t 자료형
    - sizeof 연산자가 반환하는 값의 자료형
  - ptrdiff_t 자료형 (잘 몰라도 됨. 포인터의 차는 나중에 학습함)
    - 메모리 주소에서 다른 주소를 뺄 때의 차(difference)를 의미하는 자료형

1. 자료형

■ 기본 자료형

▪ 실수 자료형

- float, double, long double 자료형 세 가지를 제공
- 컴파일러에 따라 long double 형이 없거나 double 형과 동일할 수 있음

표 2-7 float, double, long double 자료형의 표현 가능 범위

종류	사용 메모리 공간 크기	범위
float	4바이트	1.175494e-38 ~ 3.402823e+38
double	8바이트	2.225074e-308 ~ 1.797693e+308
long double	16바이트	3.362103e-4932 ~ 1.189731e+4932

기본 자료형

- 실수를 메모리에 저장하는 방법
  - 부동소수점 저장 방법 사용

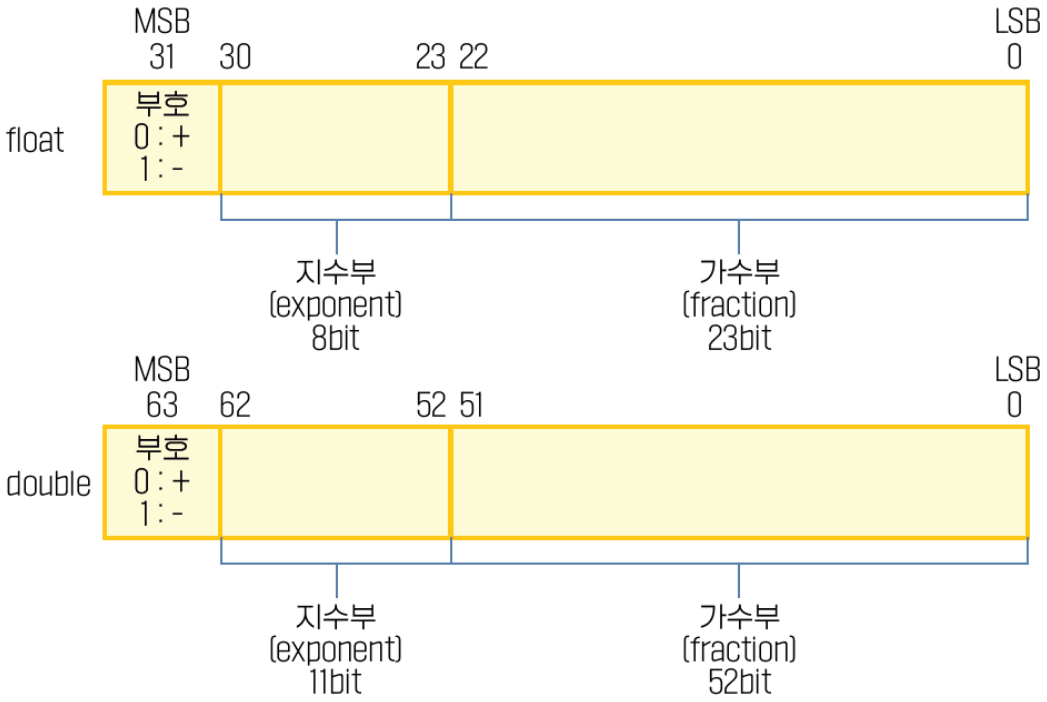

그림 2-7 단정밀도 실수와 배정밀도 실수를 저장하는 메모리

기본 자료형

실수 계산의 오차 문제

코드 2-9 FloatError.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       float f = 0.01f * 0.01f; // C 언어에서는 곱셈을 *로 표시함
6       printf("f = %f\n", f);
7       printf("f = %.20f\n", f);
8       return 0;
9   }
```

0.01과 0.01을 곱해서 결과를 f에 저장

f에 저장된 0.01 x 0.01의 결괏값을 소수점 이하 여섯 자리까지 출력

f에 저장된 0.01 x 0.01의 결괏값을 소수점 이십 번째 자리까지 출력

<실행 결과>

```
f = 0.000100
f = 0.00009999999747378752
```

기본 자료형

- 실수 계산의 오차 문제
  - 똑같은 코드를 double로 변경해서 작성

코드 2-10 DoubleError.c

```c
#include <stdio.h>

int main(void)
{
    double f = 0.01 * 0.01; // C 언어에서는 곱셈을 * 로 표시함
    printf("f = %f\n", f);
    printf("f = %.20f\n", f);
    return 0;
}
```

double 형 변수에 저장

<실행 결과>

```
f = 0.000100
f = 0.00010000000000000
```

- 웬만하면 double 형 사용 권장

02

변수

2. 변수

- 프로그램이 실행하는 동안 변경되는 값을 저장

- C 언어 수업의 성적을 계산(자신이 컴퓨터라고 가정)
- 중간고사, 기말고사, 과제 비율은 각각 35, 40, 25%, 만점은 100점

```
학생_점수 = 중간고사_점수 * 0.35 + 기말고사_점수 * 0.40 + 과제_점수 * 0.25
```

- 학생 A : 중간, 기말, 과제 점수를 각각 80, 70, 80점
- 학생 B : 90, 70, 80점

학생 이름	중간고사 점수	기말고사 점수	과제 점수	최종 점수
~~A~~ B	~~80 (28)~~ 90 (31.5)	~~70 (28)~~ 70 (28)	~~80 (20)~~ 80 (20)	~~76~~ 79.5

그림 2-8 A와 B의 과제 점수 계산 과정

2. 변수

변수의 세 가지 용도

- 메모리 공간
- 변숫값
- 변수 이름

2. 변수

변수 이름 짓는 방법

- 첫 번째 글자는 영문 알파벳
- 예외적으로 밑줄 문자(_)는 첫 번째 글자로 사용할 수 있지만, 특수 목적으로 예약
- 두 번째 글자부터는 밑줄 문자, 영문 알파벳, 숫자 사용 가능
- 밑줄 문자를 제외한 다른 특수 기호 사용 불가
- 대/소문자 다르게 구분
- 여러 단어를 사용하는 경우 밑줄 문자로 단어를 분리하거나 카멜 케이스 방식 사용
- C 언어의 키워드는 식별자 이름으로 사용 불가

변수 이름 짓는 방법

표 2-8 ANSI와 C99 버전 C 언어의 키워드(C99 버전은 따로 표기)

키워드					
auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	restrict (C99)	return	short	signed	sizeof
static	struct	switch	typedef	union	unsigned
void	volatile	while	_Bool (C99)	_Complex (C99)	_Imaginary (C99)

표 2-9 잘못 사용된 식별자와 이유 설명

잘못 사용된 식별자	설명
1a	숫자로 시작하면 오류가 발생한다. a1이 맞는 표현이다.
a:b	사용할 수 없는 특수 기호가 포함된다.
number-student	밑줄 문자(_) 대신 '-' 문자를 사용하면 오류가 발생한다. number_student라고 쓴다.
auto	키워드를 사용한다.

03

변수 선언, 대입문, 초기화

3. 변수 선언, 대입문, 초기화

변수 선언의 의미

- 변수가 사용하는 메모리 공간을 확보
- 변수 이름을 코드에 알리는 것

3. 변수 선언, 대입문, 초기화

변수 선언 방법

- 자료형과 이름을 명시

```
자료형 변수_이름;
```

- int와 double 형 변수를 선언

코드 2-11

```
int n;
double d;
```

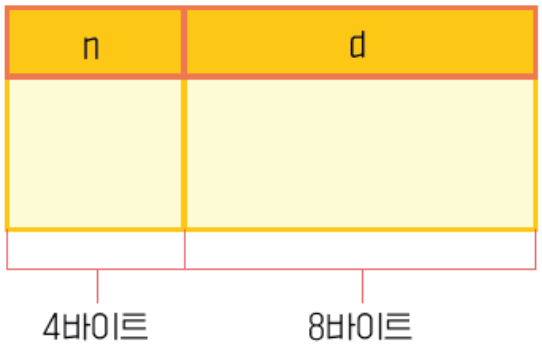

그림 2-9 메모리에 할당된 n과 d 변수

3. 변수 선언, 대입문, 초기화

변수 선언 방법

- 같은 자료형으로 여러 개 변수 선언

```
자료형 변수_이름1, 변수_이름2, 변수_이름3;
```

- int 형 변수 세 개 선언

코드 2-12

```
int n1, n2, n3;
```

쓰레기 값

- 변수를 선언할 때 할당받은 변수의 메모리 공간에 무의미한 값

3. 변수 선언, 대입문, 초기화

변수 초기화와 대입문

- 선언한 변수 초기화

코드 2-13

```
n = 3;
d = 2.3;
```

- 대입문 문법

```
변수 = 표현식
```

표 2-10 표현식의 종류와 평가 방법

표현식의 종류	평가 방법
리터럴 상수	상숫값
변수	변숫값(변수 공간에 있는 값)
연산식(계산 연산자, 대입 연산자, sizeof 등을 사용하는 코드)	연산 결괏값
값을 반환하는 함수	함수를 실행한 뒤 반환되는 결괏값

변수 초기화와 대입문

- 대입문 동작

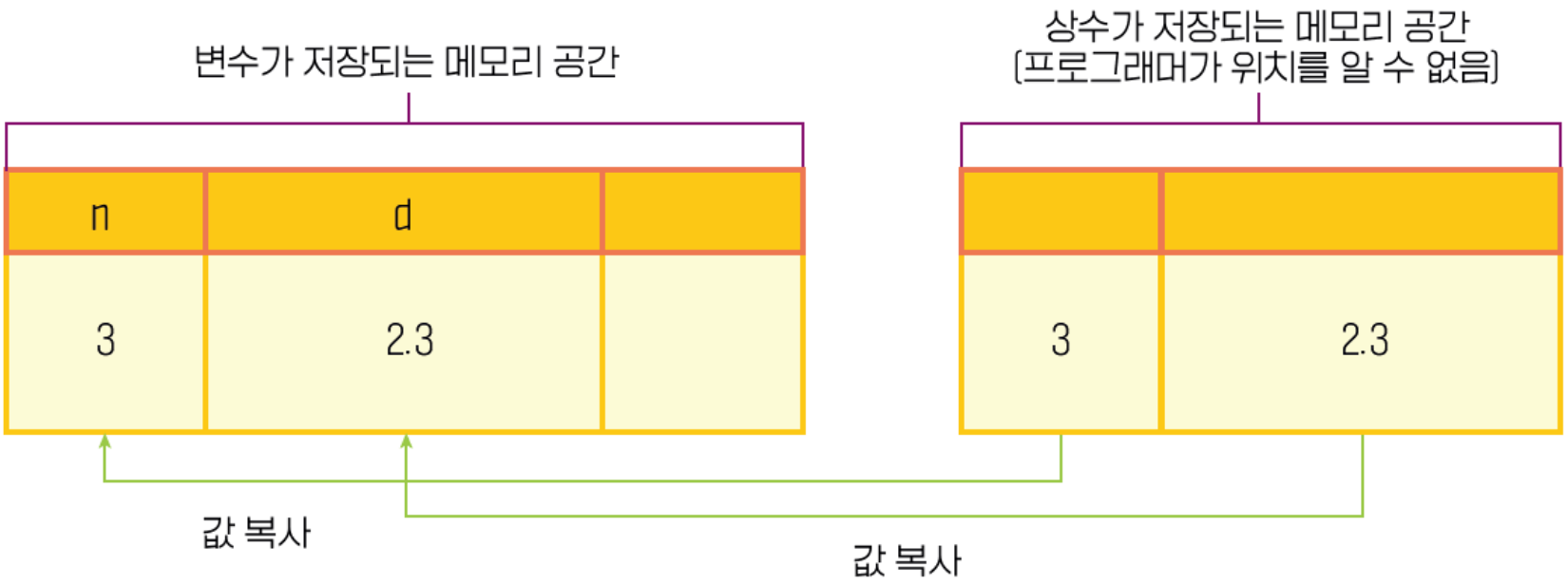

그림 2-10 대입문의 동작

3. 변수 선언, 대입문, 초기화

■ 변수 초기화와 대입문

- 변수를 초기화시키기 위해 다른 변수 사용

코드 2-14

```
n1 = n;
```

그림 2-11 변수의 메모리 공간에 복사

3. 변수 선언, 대입문, 초기화

변수 선언과 초기화

- 변수를 선언하면서 값을 지정해서 초기화

```
자료형 변수 = 값;
```

- 코드 2-14를 변수 선언과 동시에 초기화시키는 코드

코드 2-16

```
int n = 3;
double d = 2.3;
```

- 피연산자로 변수를 취하고 주솟값을 반환하는 단항 연산자

```
&변수_이름
```

3. 변수 선언, 대입문, 초기화

■ 주소 연산자로 변수의 메모리 주소 알아내기

- 변수들을 선언하고 주솟값 출력

코드 2-17 AddressOf.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       char ch = 'a';
6       int n = 1;
7       double d = 1.0;
8
9       printf("address of ch = %p\n", &ch);
10      printf("address of n = %p\n", &n);
11      printf("address of d = %p\n", &d);
12      return 0;
13  }
```

<실행 결과>

```
address of ch = 000000075FEFF830
address of n = 000000075FEFF834
address of d = 000000075FEFF838
```

04

표준 출력 함수 printf()

4. 표준 출력 함수 printf()

- 다양한 자료형의 값들을 서식에 맞춰 화면에 출력하는 표준 C 라이브러리의 함수
- 반드시 stdio.h 헤더 파일(header file)을 포함

코드 2-18

```
#include <stdio.h>  // 헤더 파일

int printf("출력할 문자열 또는 서식 문자열"[, 값1, 값2... ]);
```

문자열 출력

- printf()의 가장 기본적인 사용법은 문자열을 출력하는 것

```
printf("My first C program");
```

4. 표준 출력 함수 printf()

■ 문자열 출력

코드 2-19 Printf2.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       printf("My\tfirst C\nprogram\n");
6       printf("My ");
7       printf("second");
8       printf(" string.");
9       return 0;
10  }
```

My와 first 사이에 탭으로 띄어쓰기하고, C 다음에 줄 바꿔서 program을 출력한 후 다시 줄바꿈

줄바꿈 문자가 문자열에 포함되지 않으면 printf() 함수를 여러 번 호출해도 같은 줄에 문자열이 계속 출력됨

<실행 결과>

```
My      first C
program
My second string.
```

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

- 다이어트에 필요한 음식 100g당 칼로리 출력

표 2-11 음식 별 칼로리

음식	100g 기준 칼로리(단위: kcal)	음식	100g 기준 칼로리(단위: kcal)
steamed rice	147.62	banana	93.0
fried egg	193.48	boiled egg	136.0

```
Calories of steamed rice (100g): 147.62 kcal
Calories of banana (100g): 93.0 kcal
Calories of fried egg (100g): 193.48 kcal
Calories of boiled egg (100g): 136.0 kcal
```

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

- 다이어트에 필요한 음식 100g당 칼로리 - 문자열 출력 코드

코드 2-20

```
1    printf("Calories of steamed rice (100g): 147.62 kcal\n");
2    printf("Calories of banana (100g): 93.0 kcal\n");
3    printf("Calories of fried egg (100g): 193.48 kcal\n");
4    printf("Calories of boiled egg (100g): 136.0 kcal\n");
```

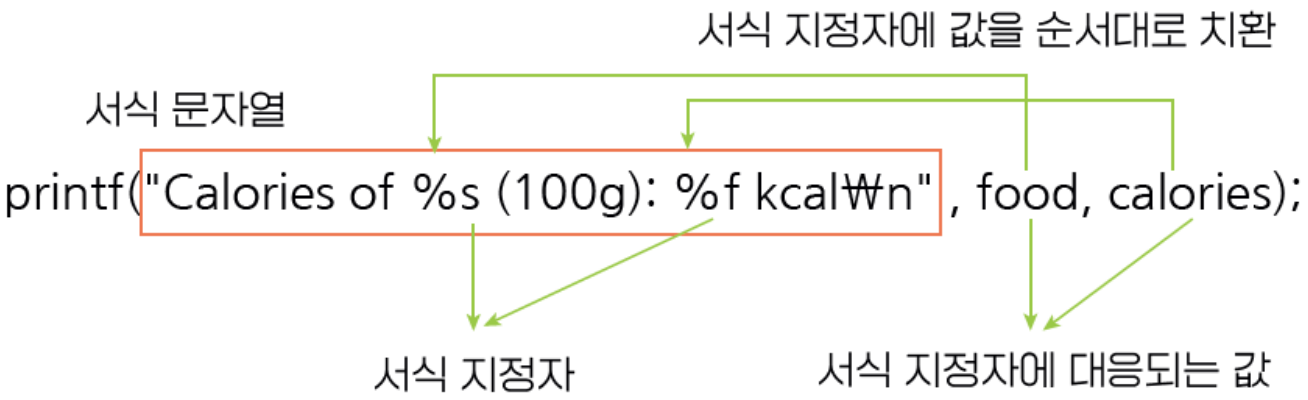

그림 2-12 printf() 함수

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

- 서식 지정자 사용법 - 필수 요소

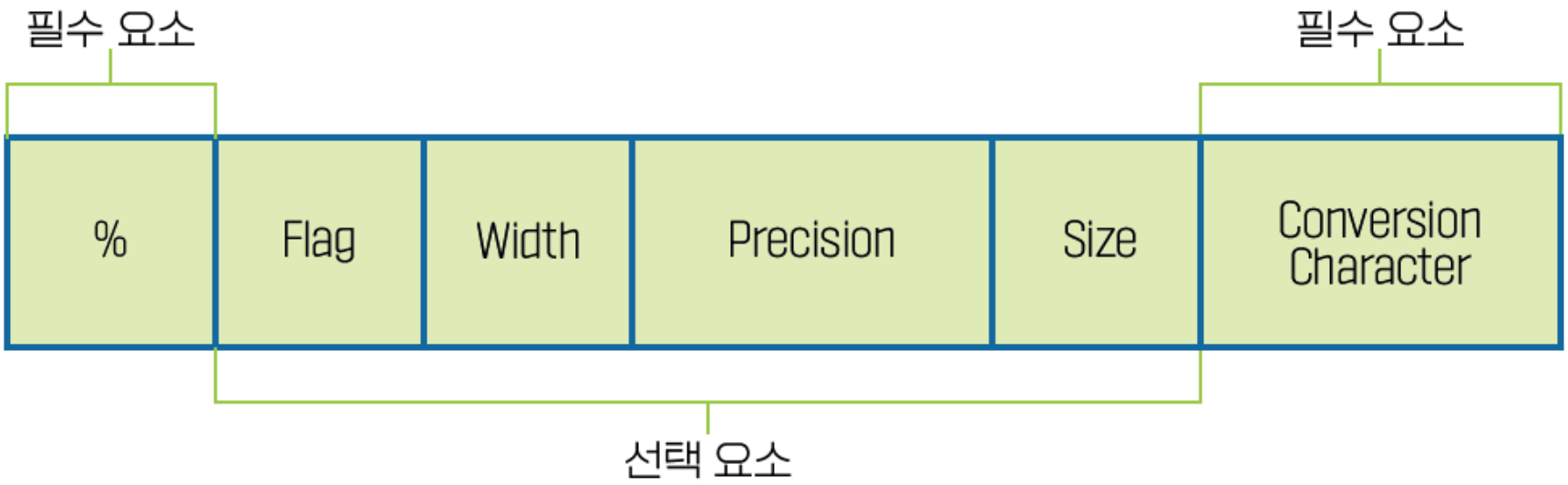

그림 2-13 변환 문자의 필수 요소와 선택 요소

4. 표준 출력 함수 printf( )

서식에 맞춘 값 출력

서식 지정자 사용법 - 필수 요소

표 2-12 서식 지정자와 설명

서식 지정자	값의 자료형	설명	화면 출력 예시
%d 또는 %i	signed int	정수를 10진수로 출력한다. 음수는 부호가 출력된다.	-10
%u	unsigned int	0 또는 양의 정수를 10진수로 출력한다.	10000000
%x 또는 %X	unsigned int	0 또는 양의 정수를 16진수로 출력한다. 16진수를 나타내는 0x나 0X는 출력되지 않는다. %x는 a~f를 소문자로 %X는 A~F를 대문자로 출력한다.	5ac0
%f	float double	소수점 이하 6자리까지 출력한다. double과 float 자료형 값을 출력할 때 사용 가능하다.	23.122334
%e 또는 %E	float double	실수를 실수E지수 형식으로 출력한다. %e는 실수e지수, %E는 실수E지수 형태로 표시한다.	2.3E5
%g 또는 %G	float double	%g는 컴파일러가 %f 또는 %e 중 한 가지를 선택해서 출력한다. %G는 %F 또는 %E 중 한 가지를 선택한다.	2.3E-4 또는 2.3E8
%c	int	int를 unsigned char 형으로 형 변환 후 문자 형태로 출력한다.	a
%s	char*	문자열을 출력한다.	hello world
%p	void*	컴파일러마다 출력 내용이 다를 수 있다. 보통 주소를 출력한다.	000000B4D2EFFD80
%%	없음	% 글자를 출력한다.	%

※ Size를 사용해 long int, size_t 등을 출력하는 코드

➔ 코드 2-21
➔ 실행 결과

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

서식 지정자 사용법 - 선택 요소

- Width

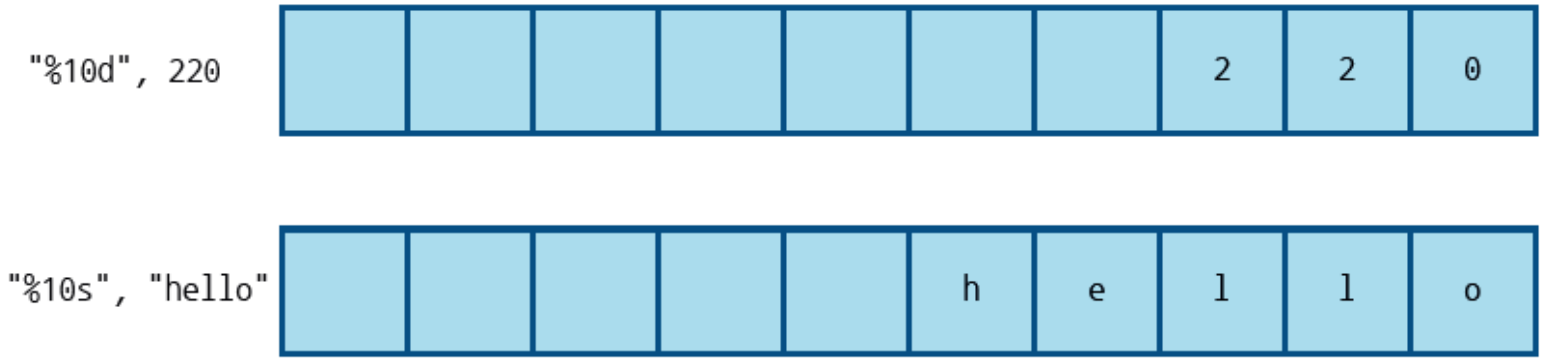

그림 2-14 선택 요소 Width 예시 (1)

그림 2-15 선택 요소 Width 예시 (2)

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

서식 지정자 사용법 - 선택 요소

- Flag

표 2-13 Flag 값 설명

Flag 문자	설명
-	- 는 항상 Width와 함께 사용된다. 주어진 Width가 값의 길이보다 클 때 왼쪽으로 정렬시키고 남는 칸수만큼 띄어쓰기한다.
0	0도 Width와 함께 쓰이고 숫자들을 출력할 때 유효하다. - Flag와 함께 쓰일 때는 왼쪽에 채울 내용이 없어지므로 의미가 없다. 출력하는 값의 길이가 Width보다 작을 때 남는 공간을 0으로 채운다.
+	양수 또는 음수를 출력할 때 '+' 또는 '-' 부호를 항상 붙인다.

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

서식 지정자 사용법 - 선택 요소

- Precision
  - 변환 문자가 d, i, u, x, X일 때 출력해야 하는 최소한의 숫자 개수
  - 변환 문자가 f, e, E일 때 출력해야 하는 소수점 이하 숫자 개수
  - 변환 문자가 g, G일 때 무조건 출력해야 하는 유효 숫자(소수점을 기준으로 앞뒤에 있는 숫자)의 개수
  - 문자열을 출력할 때 출력해야 하는 최대 문자 개수

4. 표준 출력 함수 printf()

서식에 맞춘 값 출력

- 서식 지정자 사용법 - 선택 요소

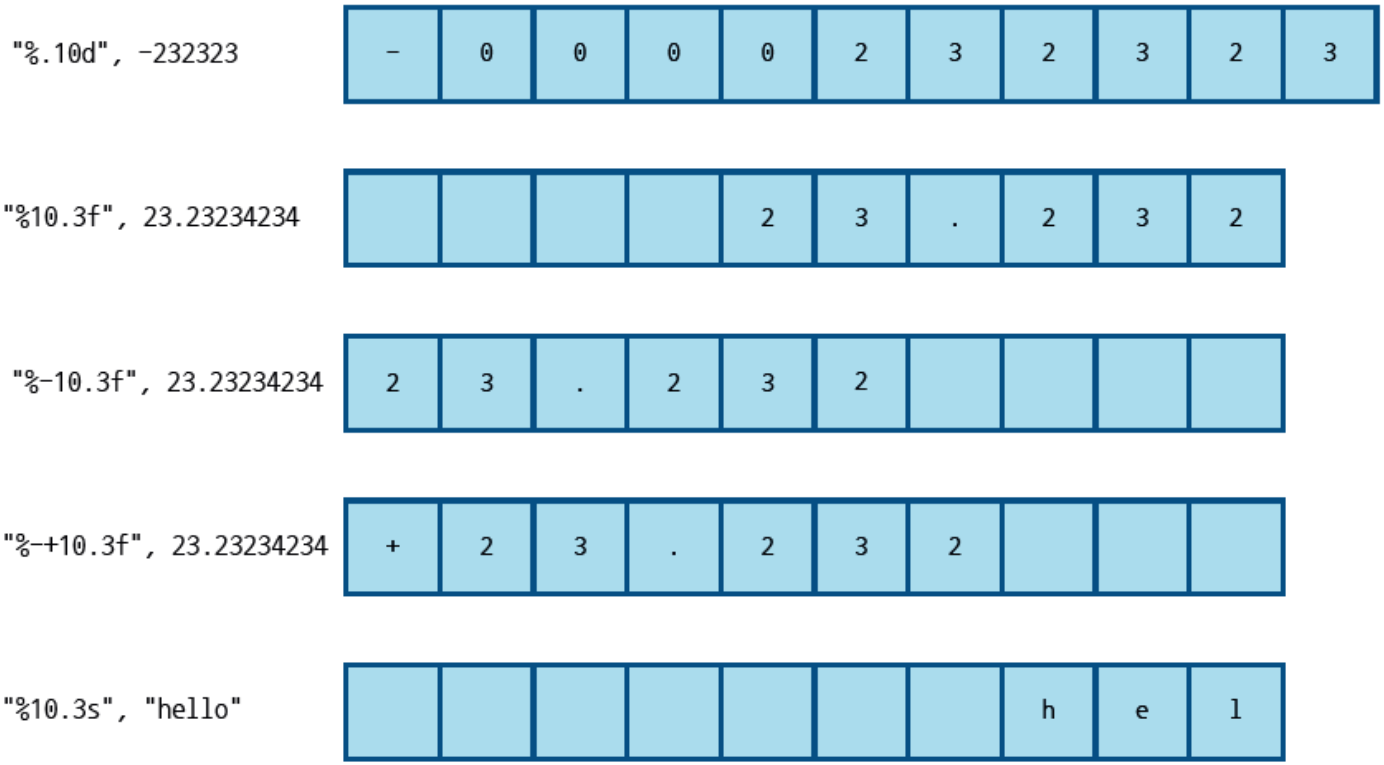

그림 2-16 Flag, Width, Precision 동작

※ Flag, Width, Precision 사용 코드

➔ 코드 2-22
➔ 실행 결과

4. 표준 출력 함수 printf()

■ 서식에 맞춘 값 출력

▪ 서식 지정자 사용법 - 선택 요소

- Size

  – long int, long long int, long double, size_t 등을 명시하기 위해 사용

표 2-14 Size 요소

Size	함께 사용 가능 변환 문자	설명
l	d, i, u, x	long int, unsigned long int를 출력할 때 사용한다(예 : %ld, %lu).
ll	d, i, u, x	long long int, unsigned long long int를 출력할 때 사용한다(예 : %lld).
L	f, e, E, g, G	long double을 출력할 때 사용한다(예 : %Lf).
z	d, i, u, x	주로 size_t 자료형을 출력할 때 %zu 형태로 사용한다.
t	d, i, u, x	ptrdiff_t 자료형을 출력할 때 %td 형태로 사용한다.

※ Size로 long int, size_t 등을 출력하는 코드

➔ 코드 2-23
➔ 실행 결과

05

형 변환

5. 형 변환

자동 형 변환

- 컴파일러가 알아서 적합한 자료형으로 변환하는 것
- 암묵적 형 변환이라고도 함

코드 2-24

```
int n = 2.3;        // double 자료형 2.3을 int 형 n에 저장
double d = 2;       // int형 2를 double 형 변수에 저장
```

- C 언어에서 자동 형 변환은 정수 ↔ 실수 자료형에서 발생
- 정수 ↔ 정수, 실수 ↔ 실수 사이에서도 발생

5. 형 변환

■ 자동 형 변환

- char(1) → short(2) → int(4), long(4) → long long(8)
- unsigned char(1) → unsigned short(2) → unsigned int(4), unsigned long(4) → unsigned → long long(8)
- float(4) → double(8)

※ 자동 형 변환이 일어났을 때 문제 없는 코드

➔ 코드 2-25
➔ 실행 결과

5. 형 변환

자동 형 변환

- A → B 방향으로 형 변환할 때 문제될 수 있는 대표적인 것

표 2-15 형 변환할 때 문제점

A	B	설명
큰 범위 signed 정수형	작은 범위 signed 정수형	B가 표현할 수 있는 범위에 포함되면 값을 유지한다. 유지하지 않으면 문제가 발생한다.
큰 범위 unsigned 정수형	작은 범위 unsigned 정수형	B가 표현할 수 있는 범위에 포함되면 값을 유지한다. 유지하지 않으면 문제가 발생한다.
signed 정수형	unsigned 정수형	B가 표현할 수 있는 범위에 있으면 값을 유지한다. 아니면 값 손실 문제가 발생한다.
unsigned 정수형	signed 정수형	B가 표현할 수 있는 범위에 있으면 값을 유지한다. 아니면 값 손실 문제가 발생한다.
double	float	범위 내에 포함될 때 값을 유지할 가능성이 높다. 하지만 실수를 메모리에 저장하는 방법 때문에 보장하지 못한다. 최대한 근접한 값으로 저장한다. 범위 밖이라면 문제가 발생한다.
int, long	float	실수를 메모리에 저장하는 방법 때문에 소실되는 값이 있을 수 있다.

5. 형 변환

자동 형 변환

- GCC에서 자세한 경고(warning)를 확인하고 싶다면 변환 경고 옵션("-Wconversion")을 지정
- 가능하면 "-Wall" 옵션까지 붙여 모든 경고를 활성화하는 것이 좋음

```
gcc -Wall -Wconversion Convert2.c
```

※ 컴파일러 경고 예시 코드

➔ 코드 2-26
➔ 컴파일 결과
➔ 실행 결과

5. 형 변환

명시적 형 변환

- 프로그래머가 특정 자료형으로의 변환을 컴파일러에 직접 요청하는 것
- 변환할 표현식 앞에 형 변환 연산자(cast operator)를 표시
- 형 변환 연산자는 변환 목표 자료형을 괄호로 감싸서 표시

```
(자료형) 표현식  // 명시적 형 변환 방법
```

- 코드 2-25를 명시적 형 변환으로 변경

코드 2-27

```
int n = (int) 2.3;      // 자동 형 변환 코드 int n = 2.3;
double d = (double) 2; // 자동 형 변환 코드 double d = 2;
```

- 임시로 값을 다른 자료형으로 변경해야 할 때 유용

```
int n = 65;
```

5. 형 변환

■ 명시적 형 변환

코드 2-28 Convert3.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       int n = 65;
6
7       printf("n = %d\n", n); // n을 정수로 출력. 65가 출력됨
8       printf("n = %f\n", n); // n을 실수로 출력. 0.000000이 출력됨
9       printf("n = %f\n", (float) n); // n을 실수로 출력. 65.000000이 출력됨
10      return 0;
11  }
```

int 형 변수 n을 정수로 출력. 65가 출력

int 형 변수 n을 실수로 출력하여 65.00이 아니라 0.00이 출력됨
정수와 실수의 메모리 저장 방법이 다른데, 방법 상관없이 사용해서 문제가 발생

강제 형 변환. n을 실수 형태로 변환해서 임시 실숫값을 구하고 그 값을 출력
65.00이 올바르게 출력됨

<실행 결과>

```
n = 65
n = 0.000000
n = 65.000000
```

06

typedef – 자료형 재정의

6. typedef – 자료형 재정의

- 자료형에 새로운 이름(alias) 붙이기 가능

```
typedef 기존_자료형_이름 새로운_자료형_이름;
```

- int를 INT로, long long int를 LLINT로 변경

```
typedef int INT;
typedef long long int LLINT;
```

- 새로 만든 자료형으로 변수를 만들고 초기화

```
INT a = 3;
LLINt b = 10ll;
```

- 형 변환을 할 때 사용할 수도 있고, sizeof() 연산자에도 전달 가능

```
a = (INT) b;
a = sizeof(INT);
```

6. typedef – 자료형 재정의

- typedef 사용 코드

코드 2-29 Typedef1.c

```c
#include <stdio.h>

int main(void)
{
    typedef int INT;
    typedef long long int LLINT;

    INT a = 3;
    LLINT b = 10ll;
    a = (INT) b;
    printf("a = %d\n", a);
    printf("b = %lld\n", b);
    printf("size of INT = %zu\n", sizeof(INT));
    return 0;
}
```

<실행 결과>

```
a = 10
b = 10
size of INT = 4
```

07

심볼릭 상수

7. 심볼릭 상수

- 값이 변경되지 않는 상수를 사용해야 할 경우

  - 원의 면적이나 둘레를 계산하는 프로그램에서 사용하는 원주율을 나타내는 π를 소수점 이하 여섯째 자리까지 표현했을 때의 값 3.141592
  - 파운드를 그램으로 변환하는 프로그램을 작성할 때 1파운드는 453.592그램
  - 섭씨 0도 1기압일 때의 공기 중 음속을 표현하는 331.5m/sec
  - 역학 등에서 사용하는 지구의 중력 가속도 $9.80 m/s^2$
  - 두 물체 사이의 만유인력이나 로켓을 지구 밖으로 나갈 수 있는 속도를 계산할 때 필요한 중력 상수 $6.674 \times 10^{-11} N \cdot m^2/kg^2$

7. 심볼릭 상수

- 1~12월을 표현하는 캘린더(calendar) 프로그램을 작성한다고 가정
  - 1~12 대신 JANUARY, FEBRUARY, MARCH, ..., DECEMBER 등
  - 9.80 대신 GRAVITY_ACCELERATION으로 변수

코드 2-30

```
int JANUARY = 1;
int FEBRUARY = 2;
int MARCH = 3;
...
int DECEMBER = 12;
double GRAVITY_ACCELERATION = 9.80;
```

- 월을 나타내는 month라는 변수를 만들고 값을 3월로 초기화

```
int month = MARCH;
```

7. 심볼릭 상수

- 심볼릭 상수(symbolic constant)란 의미 있는 이름을 붙여서 사용하는 상수
- C 언어에서는 크게 세 가지 방법으로 심볼릭 상수 표현 가능

- 상수 변수(constant variable)
- 열거형(enumeration type)
- 매크로 상수(macro constant)

7. 심볼릭 상수

상수 변수

- 선언할 때 값을 초기화하면, 그 이후에는 코드에서 값을 변경할 수 없는 변수

```
const 자료형 변수_이름 = 초깃값;
```

- 코드 2-30을 다시 상수 변수로 선언

코드 2-31

```
const int JANUARY = 1;
const int FEBRUARY = 2;
const int MARCH = 3;
...
const int DECEMBER = 12;
const double GRAVITY_ACCELERATION = 9.80; // 중력 가속도
```

08

열거형 - enum 자료형

8. 열거형 - enum 자료형

- 열거형
  - 1~12월 까지를 나타내는 상수를 열거형으로 선언

코드 2-32 Month1.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       enum MONTH { JANUARY = 1, FEBRUARY = 2, MARCH = 3, APRIL = 4,
6                    MAY = 5, JUNE = 6, JULY = 7, AUGUST = 8, SEPTEMBER = 9,
7                    OCTOBER = 10, NOVEMBER = 11, DECEMBER = 12 };
8
9       enum MONTH month = MARCH;
10      // month = 15;
11      printf("month = %d\n", month);
12      return 0;
13  }
```

enum MONTH 자료형을 만들고 심볼릭 상수를 생성
JANUARY, FEBRUARY 등을 열거형 상수(enum 상수)라고 부름

enum MONTH 자료형으로 month 변수를 생성하고 MARCH로 초기화

month 변수의 값을 화면에 출력
%d로 서식을 지정하는 것에 주의

<실행 결과>

```
month = 3
```

8. 열거형 - enum 자료형

열거형 선언 방법

```
enum 열거형_이름 { 상수1 [=정수_표현식], 상수2 [=정수_표현식], ..., 상수N [=정수_표현식] };
```

- [=정수_표현식] 생략 가능

```
enum NUM { NUM1, NUM2, NUM3 }; // NUM1 = 0, NUM2 = 1, NUM3 = 2
```

- 값이 지정된 상수와 지정되지 않은 상수들이 섞여 있을 때

코드 2-33

```
enum COLOR = { RED, GREEN = 10, BLUE = 20 }; // RED = 0, GREEN = 10, BLUE = 20
enum NUM2 { NUM1 = 2, NUM2 = 10, NUM3, NUM4 = 20, NUM5 };
// NUM1 = 2, NUM2 = 10, NUM3 = 11, NUM4 = 20, NUM5 = 21
```

8. 열거형 - enum 자료형

열거형 선언 방법

- 열거형 상수의 초깃값은 정수 자료형의 값이나 표현식 사용 가능

```
enum NUM { NUM1 = -2, NUM2, NUM3 = 2 + 3 }; // NUM1 = -2, NUM2 = -1, NUM3 = 5
```

- enum은 int 형으로 취급

```
enum NUM { NUM1 = 2.3, NUM2, NUM3 }; // 컴파일 오류 발생
```

- enum 키워드와 열거형_이름 함께 사용

```
enum NUM num;         // 변수 선언됨
enum NUM num2, num3; // 여러 개 변수도 선언 가능
```

- typedef 사용

```
typedef enum { 상수1 [=표현식], 상수2 [=표현식], ..., 상수N [=표현식] } 자료형_이름;

자료형_이름 변수;
```

8. 열거형 - enum 자료형

열거형 선언 방법

코드 2-34 Num.c

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       typedef enum { NUM1 = 2, NUM2, NUM3 } NUM;
6       NUM num1 = NUM1;
7       NUM num2 = NUM2;
8       NUM num3 = NUM3;
9       printf("num1 = %d, num2 = %d, num3 = %d\n", num1, num2, num3);
10      return 0;
11  }
```

typedef를 이용해서 열거형 자료형 NUM 선언

NUM 자료형 변수 선언 및 초기화

변숫값 출력

<실행 결과>

```
num1 = 2, num2 = 3, num3 = 4
```

8. 열거형 - enum 자료형

열거형의 장단점

열거형의 장점

- 코드가 단순해진다.
- 프로그래머가 실수할 수 있는 여지를 줄인다.

- 장점 1
  – 코드 단순화

코드 2-35

```
enum MONTH { JANUARY = 1, FEBRUARY, MARCH, APRIL, MAY, JUNE,
             JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER };
```

- 장점 2
  – int 형 변수를 선언하는 것에 비해 실수를 줄일 가능성 높음

8. 열거형 - enum 자료형

열거형의 장단점

열거형의 단점

- 열거형은 int 자료형으로 취급된다.
- 열거형이 달라도 동일한 이름의 열거형 상수를 사용할 수 없다.

- 열거형 상수들이 int 자료형으로 취급되는 것은 단점이 되기도 함
- enum 자료형이 달라도 값을 서로 섞어 사용할 수 있고 문제(버그)가 되기도 함

8. 열거형 - enum 자료형

■ 열거형의 장단점

▪ 열거형의 단점

코드 2-36 Month2.c

```c
#include <stdio.h>

int main(void)
{
    enum MONTH { JANUARY = 1, FEBRUARY, MARCH, APRIL, MAY, JUNE,
                 JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER };
    enum MONTH2 {
        JAN = 1, FEB, MAR, APR, MAY2, JUN, JUL, AUG, SEP, OCT, NOV, DEC   };

    enum MONTH month = MARCH;
    month = JAN;
    printf("month = %d\n", month);
    return 0;
}
```

enum MONTH2를 선언. enum 자료형이 달라도 상수 이름이 겹칠 수 없음. 따라서 enum MONTH2에 MAY를 사용하면 enum MONTH의 MAY 때문에 컴파일 오류가 발생

month는 int 형이므로 JAN을 저장해도 문제 없음

<실행 결과>

```
month = 1
```