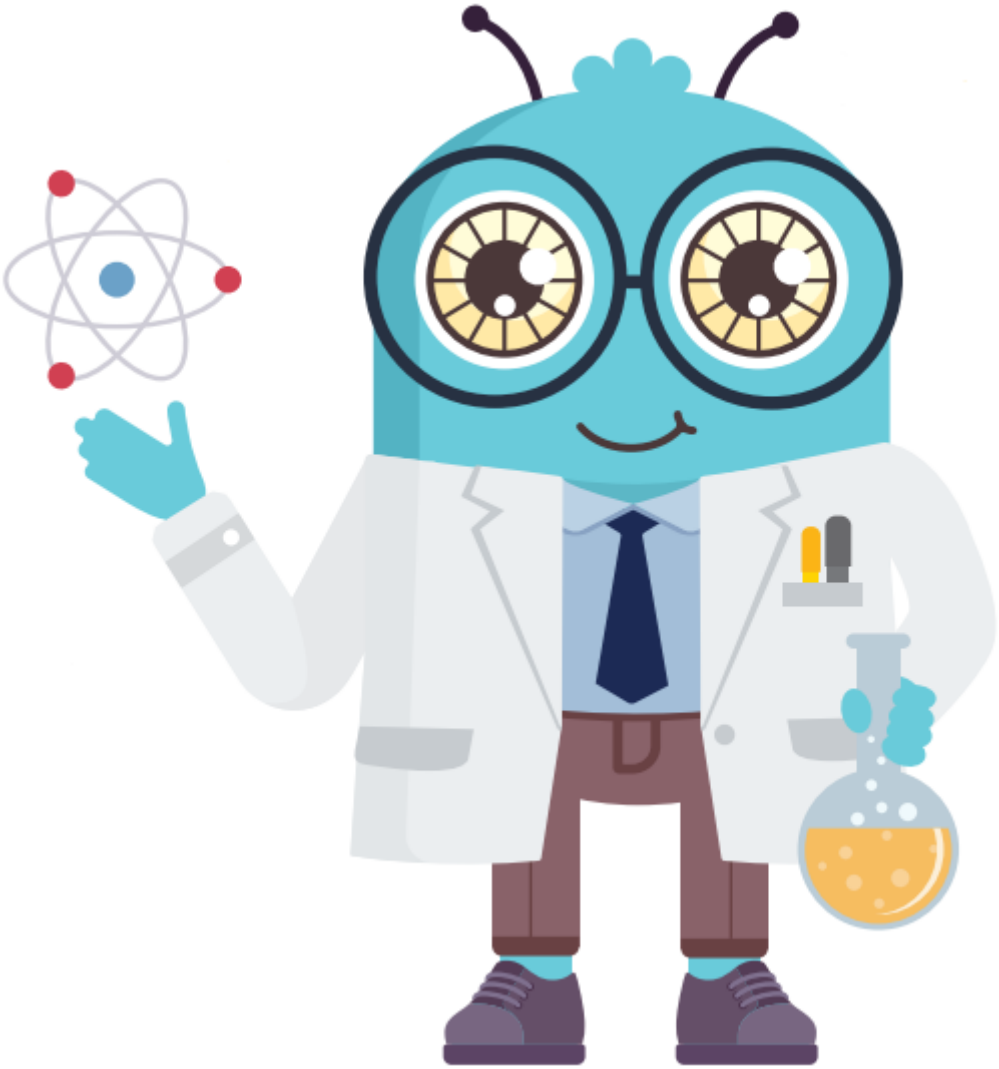

Chapter 03

printf() 함수와 데이터 형식

목차

01
printf( ) 함수의 기본 형태

1. printf( ) 함수의 기본 형태

1. printf( ) 함수의 기본 사용법(정수)

- **printf( ) 함수는 화면(모니터)에 무언가를 보여주는 역할**
- **모니터에 큰따옴표(“ ”) 안의 내용을 출력하라는 의미**

```
printf("안녕하세요?");
```

실행 결과 ▶

```
안녕하세요?
```

```
printf("100");
printf("%d", 100);
```

실행 결과 ▶

```
100
100
```

- 실행 결과 100이 두 번 출력되었지만 둘은 완전히 다른 결과
- 첫 번째 printf (“100”)의 100은 ‘숫자 100(백)’이 아닌 ‘글자 100(일영영)’
- printf( )에서 큰따옴표 안에 있는 값은 글자든 숫자 형태의 글자든 무조건 ‘글자’ 취급
- 두 번째 printf(“%d”, 100)의 결과로 나온 100은 ‘숫자 100(백)’을 의미
- 서식(%d)이 지정 된 숫자는 숫자의 의미를 그대로 지니기 때문

1. printf( ) 함수의 기본 형태

1. printf( ) 함수의 기본 사용법(정수)

기본 3-1 printf( ) 함수 사용 예 1 — 3-1.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    printf("100+100");          ----- 모두 글자로 취급한다.
06    printf("\n");
07    printf("%d", 100+100);      ----- 숫자를 계산해서 결과를 출력한다.
08    printf("\n");
09 }
```

실행 결과

```
100+100
200
```

1. printf( ) 함수의 기본 형태

1. printf( ) 함수의 기본 사용법(정수)

기본 3-2 printf( ) 함수 사용 예 2 — 3-2.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    printf("%d", 100, 200);      ----- %d는 1개, 숫자는 2개이다.
06    printf("\n");
07    printf("%d  %d", 100);       ----- %d는 2개, 숫자는 1개이다.
08    printf("\n");
09 }
```

실행 결과

```
100
100  3805219
```

1. printf( ) 함수의 기본 형태

1. printf( ) 함수의 기본 사용법(정수)

- 5행에는 %d가 하나밖에 없는데 숫자가 2개(100, 200)이고, 7행에는 %d가 2개인데 숫자는 하나(100)밖에 없음
- 숫자 2개를 출력하려면 %d 2개를 넣은 뒤 숫자 2개가 나와야 한다. 따라서 5행은 다음과 같이 고쳐야 함

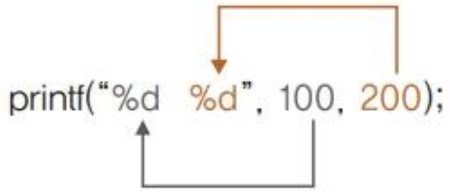

그림 3-1 서식과 숫자의 대응

SELF STUDY

100과 200을 더한 결과가 나오도록 %d를 3개 사용해서 printf( )문을 만들어보자. 또한 나눗셈의 결과도 나오게 해보자. 즉 다음과 같이 출력되게 해야 한다.

결과_ 100 + 200 = 300, 100 / 200 = 0.5

1. printf( ) 함수의 기본 형태

2. 정수 외에 자주 사용되는 서식

기본 3-3 서식을 사용한 출력 예 1 — 3-3.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    printf("%d / %d = %d", 100, 200, 0.5);
06    printf("\n");
07 }
```

----- %d가 3개, 숫자도 3개이다.

실행 결과

```
100 / 200 = 0
```

그림 3-2 서식과 숫자의 불일치

1. printf( ) 함수의 기본 형태

2. 정수 외에 자주 사용되는 서식

- [그림 3-2]와 같이, 정수 100과 200은 그대로 출력되고 마지막 0.5에 대응되는 서식이 정수를 표현하는 %d이므로 0.5에서 소수 부분이 떨어져서 0만 출력

표 3-1 printf( ) 함수의 대표적인 서식

서식	서식값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수), 정수(16진수), 정수(8진수)
%f 또는 %lf	0.5, 1.0, 3.14	실수(소수점이 있는 수)
%c	'a', 'b', 'F'	문자(한 글자이며 ' '로 둘러싸야 한다)
%s	"안녕", "abcdefg", "a"	문자열(한 글자 이상이며 " "로 둘러싸야 한다)

- 이때 문자와 문자열은 구분할 필요가 있음
- 문자는 꼭 작은따옴표(' ') 안에 한 글자만 들어 있어야 하고 문자열은 큰따옴표(" ") 안에 한 글자 이상이 들어 있어야 함

SELF STUDY

[기본 3-3]이 제대로 출력되도록 수정한 후에 빌드하고 실행해보자.

1. printf( ) 함수의 기본 형태

2. 정수 외에 자주 사용되는 서식

응용 3-4 서식을 사용한 출력 예 2 — 3-4.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    printf("%d / __1__ = __2__ \n", 100, 200, 0.5);   ---- 정수 2개와 실수 1개를 출력한다.
06    printf(" __3__ %c  \n", 'a', 'K');                ---- 문자 2개를 출력한다.
07    printf("%s %s  \n", "안녕", "c 언어");            ---- 문자열 2개를 출력한다.
08 }
```

정답 1 %d 2 %f 3 %c

실행 결과

```
100 / 200 =  0.500000
a K
안녕 C 언어
```

02
printf( ) 함수의 서식 지정

2. printf( ) 함수의 서식 지정

1. 자릿수를 맞춘 출력

기본 3-5 다양한 서식 활용 예 1 — 3-5.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    printf("%d\n", 123);
06    printf("%5d\n", 123);
07    printf("%05d\n", 123);
08
09    printf("%f\n", 123.45);
10    printf("%7.1f\n", 123.45);
11    printf("%7.3f\n", 123.45);
12
13    printf("%s\n", "Basic-C");
14    printf("%10s\n", "Basic-C");
15 }
```

05~07 ---- 정수형 서식을 활용한다.

09~11 ---- 실수형 서식을 활용한다.

13~14 ---- 문자열형 서식을 활용한다.

실행 결과

```
123
  123
00123
123.450000
  123.5
123.450
Basic-C
    Basic-C
```

2. printf( ) 함수의 서식 지정

1. 자릿수를 맞춘 출력

- 정수형 데이터를 출력하기 위한 5~7행의 서식은 [그림 3-3]과 같이 나타낼 수 있음

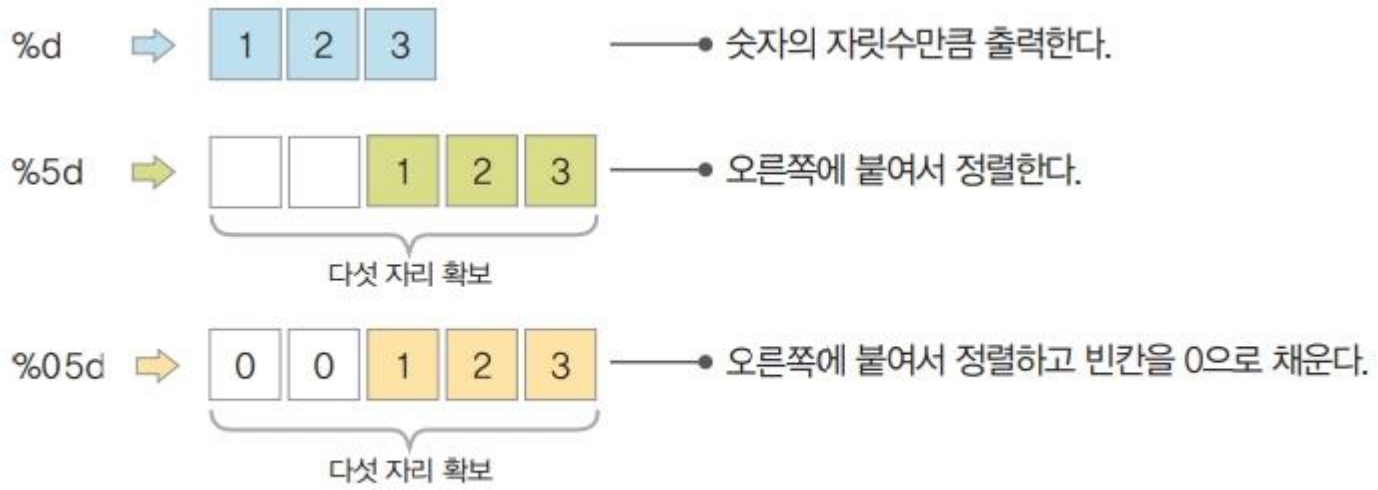

그림 3-3 정수형 데이터 서식 지정

2. printf( ) 함수의 서식 지정

1. 자릿수를 맞춘 출력

- 9~11행 실수형 데이터의 서식 지정은 [그림 3-4]와 같음
- 두 번째의 %7.1f는 소수점을 포함 한 전체 자리인 일곱 자리를 확보한 후에 소수점 아래는 한 자리만 차지한다는 의미
- 그러므로 소수점 위 정수 부분은 다섯 자리

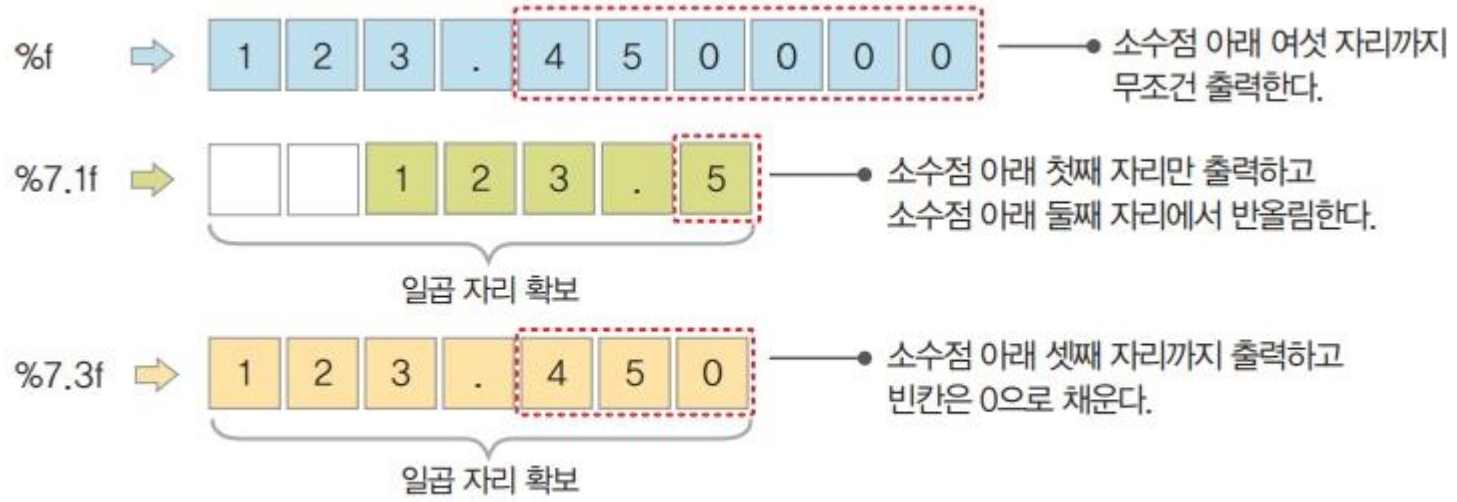

그림 3-4 실수형 데이터 서식 지정

2. printf( ) 함수의 서식 지정

1. 자릿수를 맞춘 출력

- 13, 14행 문자열형 데이터의 서식도 오른쪽에 맞춰서 출력

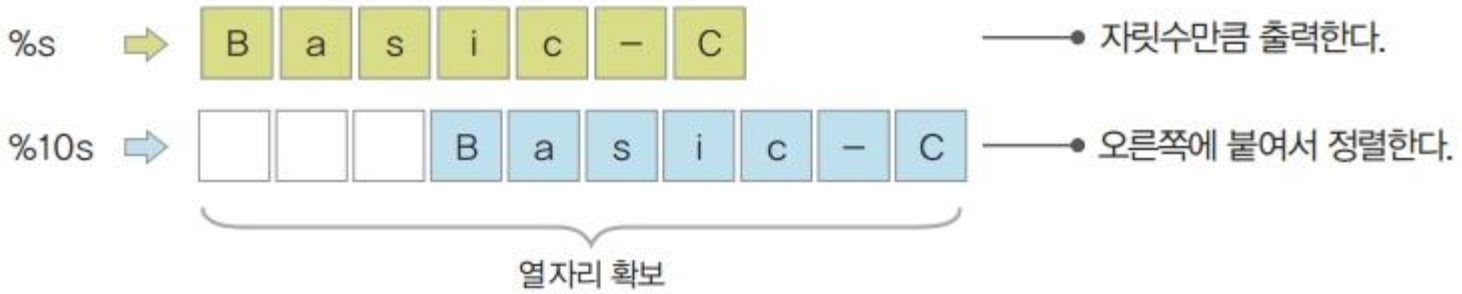

그림 3-5 문자열형 데이터 서식 지정

2. printf( ) 함수의 서식 지정

2. 다양한 기능의 서식 문자

- 앞서 줄 바꿈의 역할을 하는 ₩n의 기능을 살펴보았음
- 서식 문자의 특징은 앞에 반드시 '₩' 기호가 붙는다는 것인데 이런 문자를 탈출(escape) 문자라고도 함

표 3-2 다양한 서식 문자

서식 문자	역할	비고
\n	새로운 줄로 이동한다.	Enter를 누른 효과와 동일하다.
\t	다음 탭으로 이동한다.	Tab을 누른 효과와 동일하다.
\b	뒤로 한 칸 이동한다.	Back Space를 누른 효과와 동일하다.
\r	줄의 맨 앞으로 이동한다.	Home을 누른 효과와 동일하다.
\a	'삑' 소리를 낸다.	–
\\	\를 출력한다.	–
\'	'를 출력한다.	–
\"	"를 출력한다.	–

2. printf( ) 함수의 서식 지정

2. 다양한 기능의 서식 문자

응용 3-6 다양한 서식 활용 예 2 — 3-6.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    printf("\n줄 바꿈\n연습 \n");
06    printf("\t탭키\t연습 \n");
07    printf("이것을\r덮어씁니다 \n");
08    printf("\a\a\a삐소리 3번 \n");
09    printf("글자가 \"강조\"되는 효과 \n");
10    printf("\\\\\\ 역슬래시 세 개 출력 \n");
11 }
```

----- 컴퓨터에 따라서 소리가 한 번만 날 수도 있다.

실행 결과

```
줄바꿈
연습
        탭키    연습
덮어씁니다
삐소리 3번
글자가 "강조"되는 효과
\\\ 역슬래시 세 개 출력
```

03
변수의 이해

3. 변수의 이해

1. 변수 선언

- 요리를 하기에 앞서 그릇을 준비하듯이 C 프로그램을 작성하려면 변수 선언을 먼저 수행
- 국그릇, 밥그릇, 반찬그릇 등 다양한 그릇이 있는 것처럼 변수의 종류도 다양
- 소수점이 없는 값과 소수점이 있는 값을 담는 변수를 선언

```
int a;
float b;
```

- 두 문장으로 다음 그림과 같이 새로운 변수(그릇) 2개를 생성
- 변수(그릇)에 각각 정수와 실수를 담을 수 있음

그림 3-6 정수형 변수와 실수형 변수의 개념

3. 변수의 이해

1. 변수 선언

- 변수를 선언하는 방식은 다양
- 만약 정수형 변수 a와 b를 선언하고 싶다면 다음과 같은 방식을 사용할 수 있음
- 즉 변수의 종류가 같을 때는 변수를 개별적으로 선언해도 되고 콤마(,)를 사용하여 연속해서 선언해도 됨

```
int a;
int b;
```

==

```
int a, b;
```

3. 변수의 이해

1. 변수 선언

- 정수형 변수 a, 실수형 변수 b, 정수형 변수 c, 실수형 변수 d를 선언하는 기본 방식

❶ 가능

```
int a;
float b;
int c;
float d;
```

==

❷ 가능

```
int a, c;
float b, d;
```

≠

❸ 불가능

```
int a, float b;
int c, float d;
```

여기서 잠깐 세미콜론을 사용한 줄 표현

- 한 줄에 하나의 데이터 형식만 선언할 수 있다고 했으나 엄밀하게 말하면 '한 줄'이 아니라 '한 문장'이라고 해야 옳음
- ❷는 올바른 형식이며 세미콜론(;)으로 구분된 것은 완전히 분리 된 문장으로 취급되므로 ❶과 ❷는 같은 의미

❶

```
int a;
float b;
int c;
float d;
```

❷

```
int a;  float b;
int c;  float d;
```

2. 변수에 값을 담는 방법

▪ 기본적인 값의 대입

- 변수 a에 100을 대입하고 변수 b에 123.45를 대입하는 것을 그림으로 표현

그림 3-7 정수형 변수와 실수형 변수에 값 대입

- 100은 정수, 123.45는 실수이므로, 정수형 변수 a와 실수형 변수 b를 선언하고 변수에 값을 넣으려면 다음과 같이 나열

```
int a;
float b;
a = 100;
b = 123.45;
```

==

```
int a = 100;
float b = 123.45;
```

2. 변수에 값을 담는 방법

▪ 기본적인 값의 대입

- 만약 변수 a, b가 모두 정수형 변수라면 형식이 동일하므로 한 줄로 해결할 수 있음

```
int a;
int b;
a = 100;
b = 200;
```

==

```
int a = 100, b = 200;
```

- 변수에 값을 대입할 때는 지정된 데이터 형식만 대입해야 함

2. 변수에 값을 담는 방법

▪ 기본적인 값의 대입

기본 3-7 변수에 값 대입 예 — 3-7.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int a;          ----- 정수형 변수 a를 선언한다.
06    float b;        ----- 실수형 변수 b를 선언한다.
07
08    a = 123.45;     ----- 정수형 변수에 실수를 대입한다. → 바람직하지 않다.
09    b = 200;        ----- 실수형 변수에 정수를 대입한다. → 바람직하지 않다.
10
11    printf("a의 값 ==> %d \n", a);
12    printf("b의 값 ==> %f \n", b);
13 }
```

실행 결과

```
a의 값 ==> 123
b의 값 ==> 200.000000
```

2. 변수에 값을 담는 방법

▪ 기본적인 값의 대입

- 실행은 되었지만 8행에서 정수형 변수 a에 실수 123.45를 대입해 결과가 123만 나왔음
- 8행에서 [그림 3-8]과 같은 처리가 이루어졌기 때문

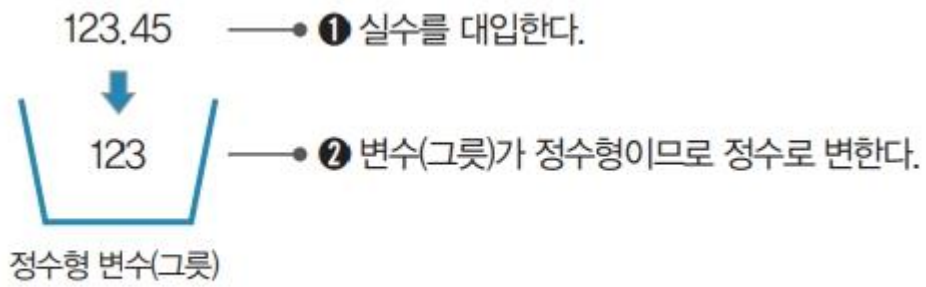

그림 3-8 정수형 변수에 실수 대입 시 처리 방식

2. 변수에 값을 담는 방법

▪ 기본적인 값의 대입

- 실수(123.45)를 대입하더라도 그것을 담는 변수(그릇)가 정수형이므로 소수점 아래가 떨어져 나가고 정수(123)만 저장되어 결국 8행의 변수 a에는 123만 들어가게 됨
- 실수형 변수(그릇)에 정수를 담으면 [그림 3-9]와 같이 처리

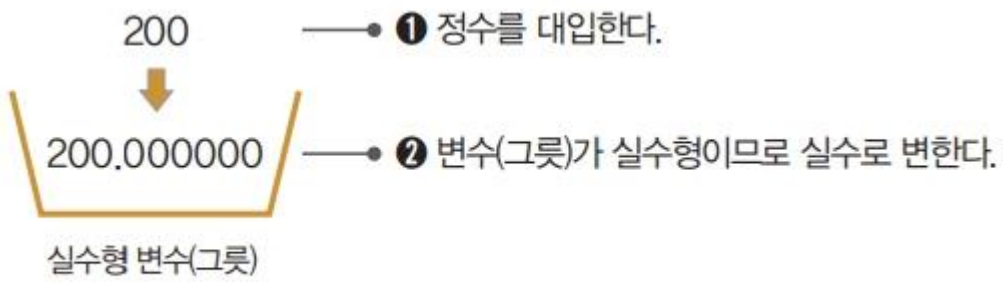

그림 3-9 실수형 변수에 정수 대입 시 처리 방식

2. 변수에 값을 담는 방법

▪ 기본적인 값의 대입

- 대입된 정수(200)가 실수(200.000000)로 변함
- 200이나 200.00이나 정수 또는 실수라는 것만 다를 뿐 값의 크기는 차이가 없으므로 문제가 생기지 않음
- 하지만 변수의 데이터 형식과 실제 값의 종류가 다른 것은 그리 바람직하지 않으므로 정수형 변수에는 정수를 대입하고 실수형 변수에는 실수를 대입해야 함
- 즉 9행은 다음과 같이 고치는 것이 바람직

```
b = 200.0;
```

2. 변수에 값을 담는 방법

▪ 다양한 값의 대입 방법

응용 3-8 변수에 변수 대입 예 1 — 3-8.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int a, b;          ----- 정수형 변수 2개를 선언한다.
06    float c, d;        ----- 실수형 변수 2개를 선언한다.
07
08    a = 100;           ----- a에 정수 100을 대입한다.
09    __1__              ----- b에 a 값을 대입한다.
10
11    c = 111.1f;        ----- c에 실수 111.1을 대입한다.
12    __2__              ----- d에 c 값을 대입한다.
13
14    printf("a, b의 값 ==> %d , %d \n", a, b);
15    printf("c, d의 값 ==> %5.1f , %5.1f \n", c, d);
16 }
```

실행 결과

```
a, b의 값 ==> 100 , 100
c, d의 값 ==> 111.1 , 111.1
```

정답 1 b = a; 2 d = c;

2. 변수에 값을 담는 방법

▪ 다양한 값의 대입 방법

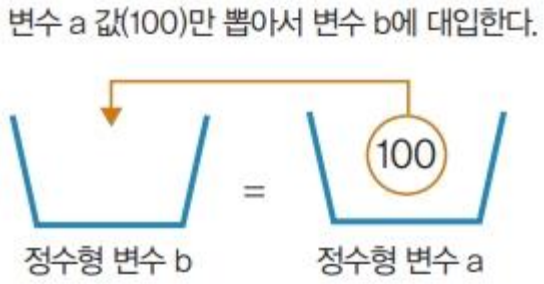

그림 3-10 변수에 변수 대입 시 처리 방식

- 8행에서 정수형 변수 a에 100을 대입
- 9행에서는 정수형 변수 b에 값 대신 a를 대입
- 이때 9행은 [그림 3-10]과 같이 처리
- 오른쪽에 있는 변수 a의 값인 100이 변수 b에 들어감
- 결국 변수 a와 b는 같은 값인 100을 가지게 되므로 14행에서 같은 값을 출력
- 15행의 실수도 동일한 결과를 출력
- 이외에도 변수에 값을 대입하는 방법은 여러 가지

3. 변수의 이해

2. 변수에 값을 담는 방법

▪ 다양한 값의 대입 방법

응용 3-9 변수에 변수 대입 예 2 — 3-9.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int  a, b, c, d;
06
07    a = 100 + 100;        ---- a에 두 수의 계산 결과를 대입한다.
08    b = a + 100;          ---- b에 변수와 수의 계산 결과를 대입한다.
09    c = a + b - 100;      ---- c에 변수와 수의 계산 결과를 대입한다.
10    __1__                 ---- d에 a, b, c의 덧셈 결과를 대입한다.
11    printf("a, b, c, d 의 값 ==> %d, %d, %d, %d \n", a, b, c, d);
12
13    __2__                 ---- a, b, c, d에 모두 100을 대입한다(한 문장으로 처리한다).
14    printf("a, b, c, d 의 값 ==> %d, %d, %d, %d \n", a, b, c, d);
15
16    a = 100;
17    a = a + 200;          ---- a에 자신의 a 값과 200을 더한 값을 다시 대입한다.
18    printf("a 의 값 ==> %d \n", a);
19 }
```

실행 결과

```
a, b , c , d 의 값 ==> 200 , 300 , 400 , 900
a, b , c , d 의 값 ==> 100 , 100 , 100 , 100
a 의 값 ==> 300
```

정답 1 d = a + b + c; 2 a = b = c = d = 100;

2. 변수에 값을 담는 방법

▪ 다양한 값의 대입 방법

- 7행에서는 연산 결과를 변수에 대입

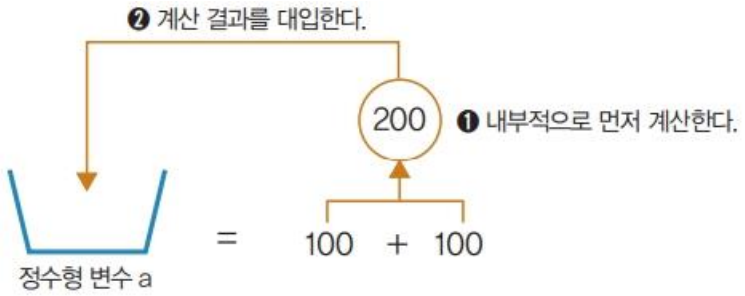

그림 3-11 숫자끼리의 계산 결과를 대입하는 방식

- 8행에서는 변수 a의 값과 100의 연산 결과를 변수 b에 대입

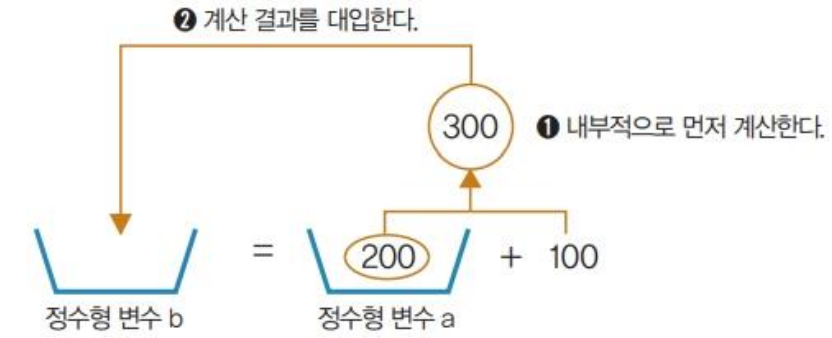

그림 3-12 변수와 숫자의 계산 결과를 대입하는 방식

2. 변수에 값을 담는 방법

▪ 다양한 값의 대입 방법

- =는 맨 뒤부터 처리 되는데 결국 13행은 다음과 같이 풀어 쓸 수 있음

```
a = b = c = d = 100;
```

==

```
d = 100;
c = d;
b = c;
a = b;
```

- 한 가지 유의할 점은 바로 전 단계(7~10행)의 a, b, c, d에 각각 200, 300, 400, 900이라는 값이 들어 있지만 그 값들은 무시하고 새로운 값으로 덮어쓴다는 것

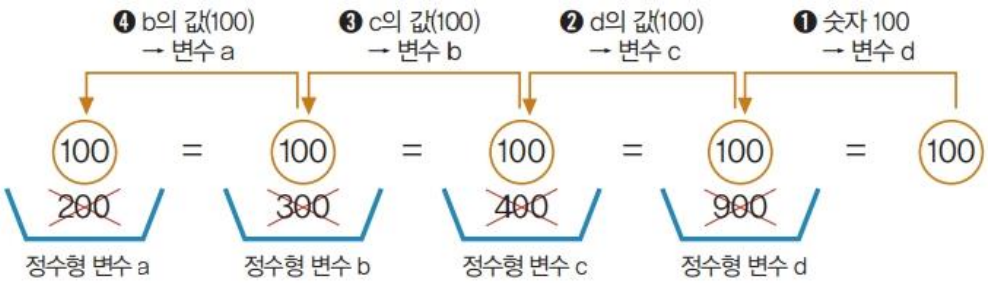

그림 3-13 연속된 값의 대입 방식

2. 변수에 값을 담는 방법

▪ 다양한 값의 대입 방법

- [그림 3-13]을 보면 대입 연산자(=)가 오른쪽에서 왼쪽 방향(←)으로 진행됨을 알 수 있음
- 변수 a, b, c, d에는 모두 100이 대입
- 17행에서는 [그림 3-14]처럼 자신의 값으로 연산을 한 후 다시 자신에게 넣음

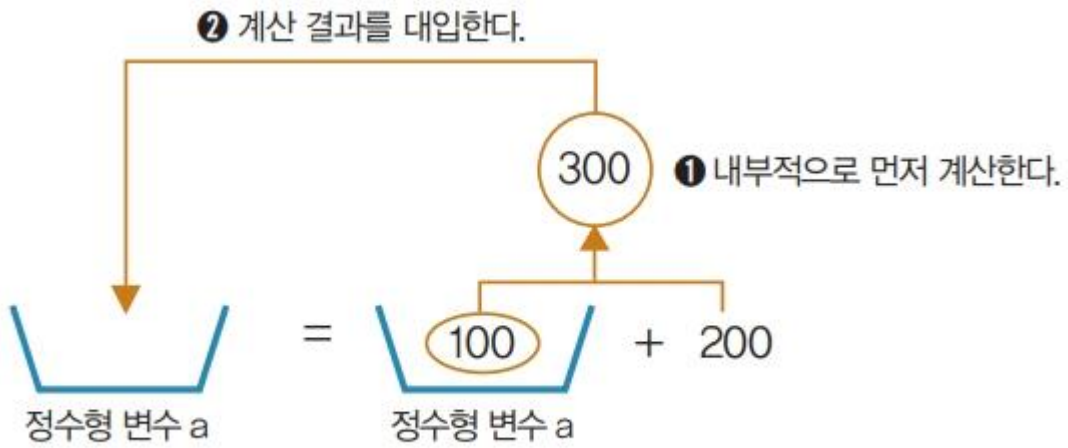

그림 3-14 자신의 값을 다시 계산 결과에 대입하는 방식

2. 변수에 값을 담는 방법

▪ 대입 연산자와 변수의 위치

- '대입 연산자(=)를 사용하면 오른쪽의 것이 왼쪽에 대입된다'는 규칙
- 대입 연산자(=)의 왼쪽에는 반드시 무엇을 담는 그릇인 변수만 온다는 것을 알 수 있음

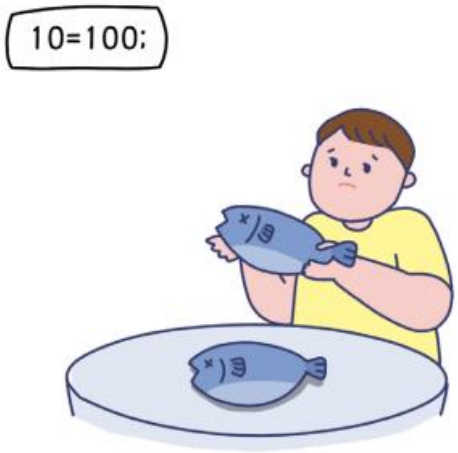

그림 3-15 값을 넣을 그릇이 없는 경우

2. 변수에 값을 담는 방법

▪ 대입 연산자와 변수의 위치

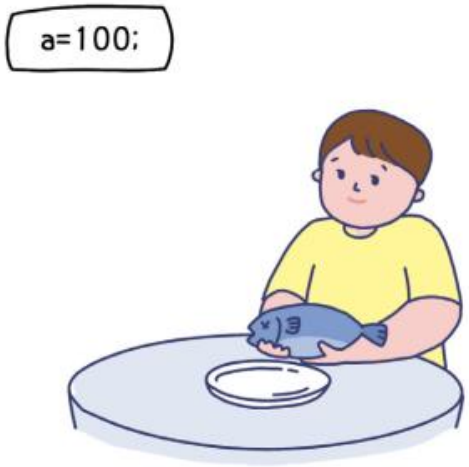

그림 3-16 값을 넣을 그릇이 있는 경우

- '대입 연산자(=)의 오른쪽에는 상수(숫자), 변수, 계산값이 모두 올 수 있다'는 규칙도 적용
- 결론적으로 대입 연산자의 왼쪽에는 변수만 올 수 있고 오른쪽에는 상수, 변수, 계산 값, 함수 등 무엇이든지 올 수 있음

04
데이터 형식과 배열

4. 데이터 형식과 배열

1. 비트, 바이트, 진수

▪ 비트

- 비트(bit)는 전기 스위치와 비슷한 개념으로, 전기 스위치에는 OFF와 ON만 있듯이 비트에도 0(OFF)과 1(ON)만 있음
- 1비트라면 전기 스위치가 1개이고 2비트라면 전기 스위치가 2개인 것

표 3-3 전기 스위치로 표현 가능한 가짓수

전기 스위치	의미	2진수(0, 1)	10진수
	꺼짐, 꺼짐	00	0
	꺼짐, 켜짐	01	1
	켜짐, 꺼짐	10	2
	켜짐, 켜짐	11	3

- 전기 스위치 2개(2비트)로 표현할 수 있는 가짓수는 4개이고 이를 2진수로 표현하면 각각 00, 01, 10, 11, 이를 10진수로 나타내면 0, 1, 2, 3

> 전기 스위치 n개로 표현할 수 있는 가짓수 = 2^n

4. 데이터 형식과 배열

1. 비트, 바이트, 진수

▪ 진수

- 10진수는 숫자 10개(0~9)로 모든 숫자를 표현
- 2진수는 숫자 2개(0, 1)로만 모든 수를 표현

표 3-4 10진수, 2진수, 16진수로 나타낸 0~15

10진수(0~9)	2진수(0, 1)	16진수(0~F)
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

4. 데이터 형식과 배열

1. 비트, 바이트, 진수

▪ 진수

- 10진수의 00은 2진수의 0000과 동일하고 10진수의 01은 2진수의 0001과 동일
- 2진수 0001에서 한 자리를 올리고 그 자리에 가장 작은 숫자(0)를 넣어 0010으로 표현
- 16진수가 필요한 이유는 2진수의 네 자리와 16진수의 한 자리가 딱 맞아떨어지기 때문

4. 데이터 형식과 배열

1. 비트, 바이트, 진수

▪ 바이트

- 비트와 더불어 C에서 가장 많이 사용되는 단위는 바이트(byte)

표 3-5 비트와 바이트의 크기에 따른 숫자의 범위

비트 수	바이트 수	표현 개수	2진수	10진수	16진수
1		2^1=2	0~1	0~1	0~1
2		2^2=4	0~11	0~3	0~3
4		2^4=16	0~1111	0~15	0~F
8	1	2^8=256	0~11111111	0~255	0~FF
16	2	2^{16}=65536	0~11111111 11111111	0~65535	0~FFFF
32	4	2^{32}=약 42억	0~…	0~약 42억	0~FFFF FFFF
64	8	2^{64}=약 1800경	0~…	0~약 1800경	0~…

2. 진수 변환 연습

▪ 10진수 변환

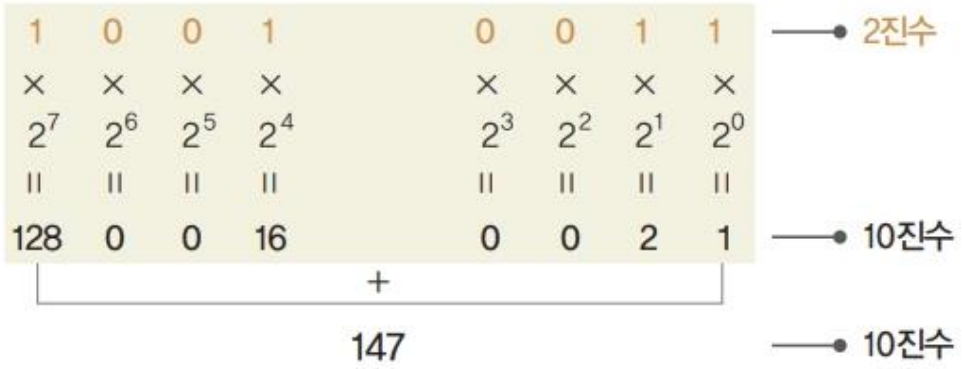

그림 3-17 2진수를 10진수로 변환하는 방법

- 2진수 1001 0011을 10진수로 변환하는 과정을 나타낸 것

2. 진수 변환 연습

▪ 10진수 변환

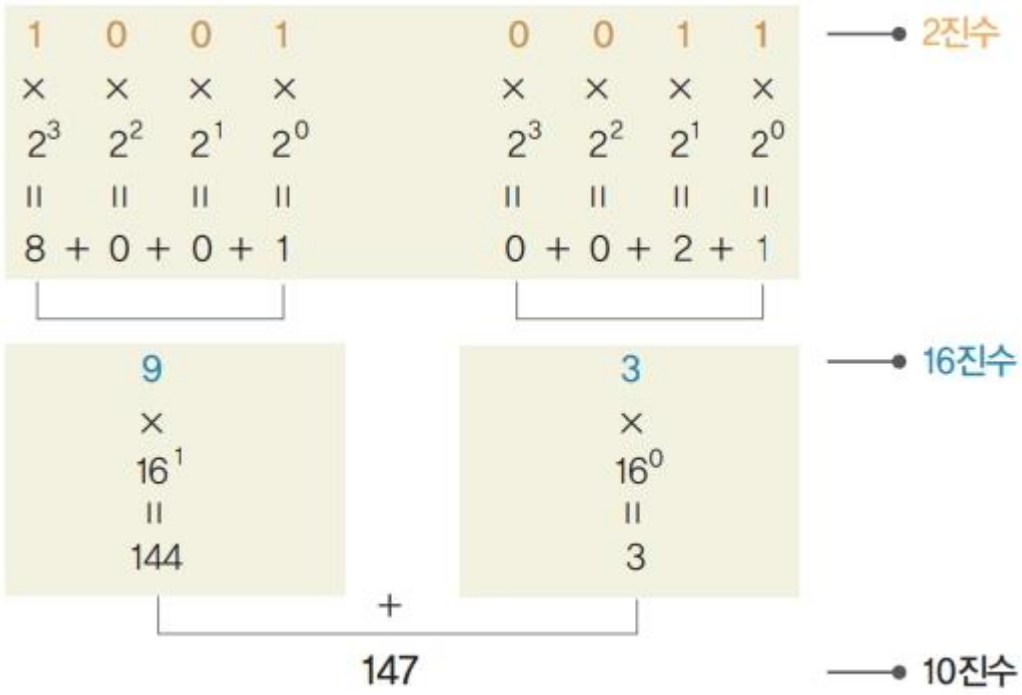

그림 3-18 2진수를 16진수로 변환한 후 10진수로 변환하는 방법

2. 진수 변환 연습

▪ 2진수 변환

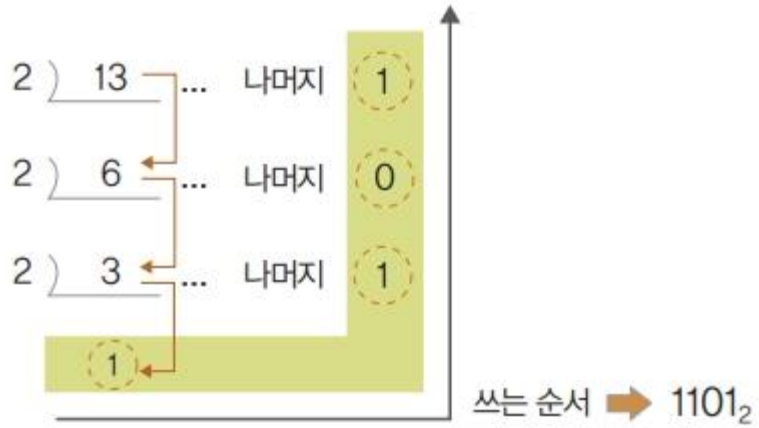

그림 3-19 10진수를 2진수로 변환하는 방법

- 처음에 13을 2로 나누면 몫은 6, 나머지는 1이고, 6을 다시 2로 나누면 몫은 3, 나머지는 0
- 다시 3을 2로 나누면 몫은 1, 나머지는 1
- 이렇게 해서 나온 마지막 몫과 나머지값들을 나열하면 2진수가 됨

2. 진수 변환 연습

▪ 2진수 변환

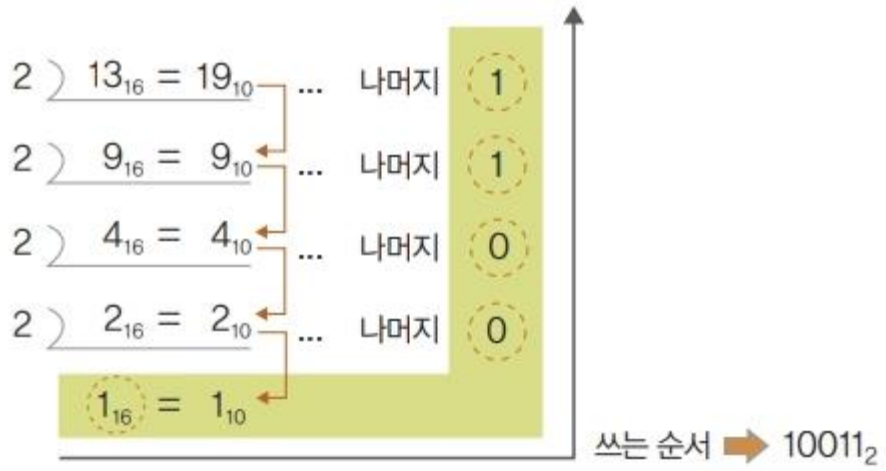

그림 3-20 16진수를 2진수로 변환하는 방법

- 13_{16}을 10진수로 바꾸면 19가 되고 이것을 2로 나누면 몫은 9, 나머지는 1
- 이후 10진수와 동일하게 계산하면 13_{16}은 2진수 100112이 됨

2. 진수 변환 연습

▪ 2진수 변환

표 3-6 16진수와 2진수

16진수	2진수	16진수	2진수
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

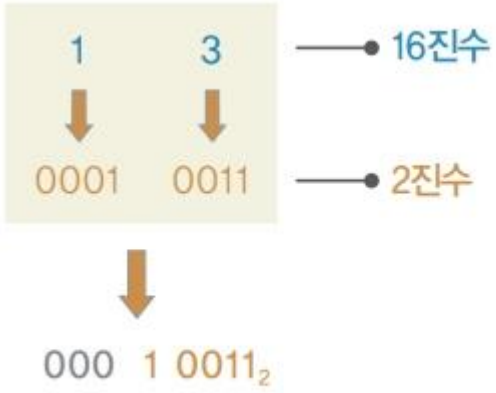

그림 3-21 16진수를 2진수로 변환하는 간편한 방법 1

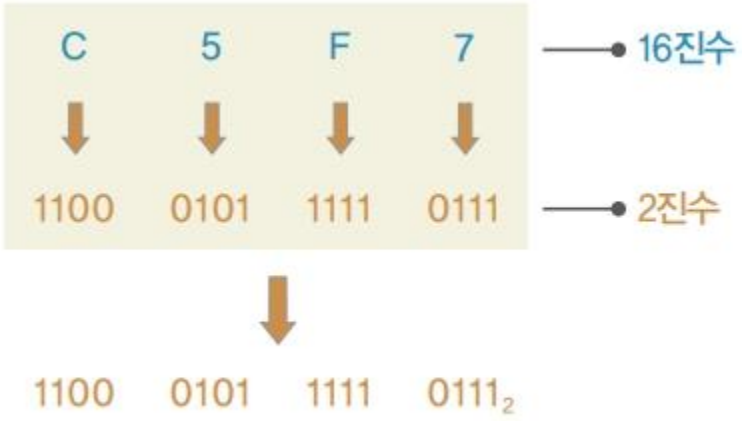

그림 3-22 16진수를 2진수로 변환하는 간편한 방법 2

3. 숫자형 데이터 형식

▪ 소수점이 없는 정수형

표 3-7 정수형 데이터 형식

데이터 형식	의미	크기	값의 범위
short	작은 정수형	2바이트	-2^{15}(–32768) ~ 2^{15}–1(32767)
unsigned short	부호 없는 작은 정수형	2바이트	0 ~ 2^{16}–1(65535)
int	정수형	4바이트	-2^{31}(약 –21억) ~ 2^{31}–1(약 21억)
unsigned int	부호 없는 정수형	4바이트	0 ~ 2^{32}–1(약 42억)
long int(또는 long)	큰 정수형	4바이트	-2^{31} ~ 2^{31}–1
unsigned long	부호 없는 큰 정수형	4바이트	0 ~ 2^{32}–1

- 정수형은 말 그대로 소수점이 없는 데이터를 입력하기 위해 사용하는 데이터 형식
- unsigned가 붙은 데이터 형식은 마이너스(-) 값이 없는 경우에 사용

3. 숫자형 데이터 형식

▪ 소수점이 없는 정수형

기본 3-10 소수점이 없는 정수형 사용 예 3-10.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int a=100, b=200;      ----- 정수형 변수 a와 b에 값을 지정한다.
06    float result;          ----- 실수형 변수 result를 선언한다.
07
08    result = a / b;        ----- a를 b로 나눈 결과를 실수형 변수 result에 대입한다.
09
10    printf("%f  \n", result);
11 }
```

실행 결과

```
0.000000
```

3. 숫자형 데이터 형식

▪ 소수점이 없는 정수형

- 100을 200으로 나눈 결과는 0.5
- 8행의 a/b는 100/200이고 100과 200은 모두 정수
- 그 결과도 실수 0.5가 아니라 소수 부분이 떨어진 정수 0 이 되어 8행의 result에 0의 실숫값인 0.000000이 저장

```
정수 +, -, *, / 정수 = 정수
정수 +, -, *, / 실수 = 실수
실수 +, -, *, / 실수 = 실수
```

3. 숫자형 데이터 형식

▪ 소수점이 있는 정수형

표 3-8 실수형 데이터 형식

데이터 형식	의미	크기	값의 범위
float	실수형	4바이트	약 -3.4×10^{38} ~ 3.4×10^{38}
double	큰 실수형	8바이트	약 -1.79×10^{308} ~ 1.79×10^{308}
long double	큰 실수형	8바이트	약 -1.79×10^{308} ~ 1.79×10^{308}

- float 형은 대개 소수 아래 일곱 자리까지의 정밀도를 나타내지만 double 형은 소 수점 아래 열여섯 자리 정도까지의 정밀도를 나타낼 수 있음

3. 숫자형 데이터 형식

▪ 소수점이 있는 정수형

기본 3-11 소수점이 있는 실수형 사용 예 — 3-11.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    float  a = 0.123456789012345678901234567f;
06    double b = 0.123456789012345678901234567;
07
08    printf("%30.25f \n", a);
09    printf("%30.25lf \n", b);
10 }
```

04행: float 형 변수 a에 정밀도 스물다섯 자리 실수를 입력한다(맨 뒤의 f는 빼도 된다).

06행: double 형 변수 b에 소수점 아래 스물다섯 자리 실수를 입력한다.

08~09행: a와 b를 소수점 아래 스물다섯 자리까지 출력하는데 float는 %f로, double은 %lf로 출력한다.

실행 결과

```
0.12345679104328155517578125
0.12345678901234567736988620
```

4. 문자형 데이터 형식

▪ 아스키코드

- 아스키코드는 컴퓨터에서 표현하는 문자(특히 키보드에 있는 영문자, 기호, 숫자 등)를 0~127에 대응한 코드

표 3-9 아스키코드

아스키코드	10진수	16진수
0 ~ 9	48 ~ 57	0x30 ~ 0x39
A ~ Z	65 ~ 90	0x41 ~ 0x5A
a ~ z	97 ~ 122	0x61 ~ 0x7A

- C에서는 숫자를 문자로도 표현

```
char ch = 'a';   ==   char ch = 97;
```

4. 문자형 데이터 형식

▪ 한 글자를 뜻하는 문자형

표 3-10 문자형 데이터 형식

데이터 형식	의미	크기	값의 범위
char	문자형 또는 정수형	1바이트	$-2^7(-128) \sim 2^7-1(127)$
unsigned char	문자형 또는 부호 없는 정수형	1바이트	$0 \sim 2^8-1(255)$

- char 형에는 문자뿐만 아니라 값의 범위에 해당하는 정수를 대입할 수 있음
- char 형을 1바이트 크기의 정수형으로 취급해도 상관없음
- char 형의 크기는 1바이트(8비트)이므로 표현할 수 있는 글자 수는 256가지이며 값의 범위는 –128~127
- 아스키코드의 0~127을 담을 수 있음

4. 문자형 데이터 형식

▪ 한 글자를 뜻하는 문자형

기본 3-12 문자형 변수 사용 예 1 — 3-12.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char a, b, c;          ----- 문자형 변수 3개를 선언한다.
06
07    a = 'A';               ----- 문자형 변수 a에 'A'를 대입한다.
08
09    printf(" %c \n", a);   ----- 문자형 변수 a를 문자형과 정수형으로 출력한다.
10    printf(" %d \n", a);
11
12    b = 'a';               ----- 문자형 변수 b에 'a'를 대입한다.
13    c = b + 5;             ----- 문자형 변수 b에 5를 더해서 문자형 변수 c에 대입한다.
14    printf(" %c \n", b);
15    printf(" %c \n", c);
16
17    c = 90;                ----- 문자형 변수 c에 90을 대입한다.
18    printf(" %c \n", c);
19 }
```

실행 결과

```
A
65
a
f
Z
```

4. 문자형 데이터 형식

▪ 한 글자를 뜻하는 문자형

응용 3-13 문자형 변수 사용 예 2 — 3-13.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    int a, b;
06    char c, d;
07
08    a = 0x41;          ---- 정수형 변수 a, b에 16진수 0x41과 0x50을 대입한다.
09    b = 0x50;
10
11    __1__              ---- 정수형 변수 b를 문자형으로 출력한다.
12
13    c = a;             ---- 문자형 변수 c에 정수형 변수 a 값을 대입한다.
14    __2__              ---- 문자형 변수 c를 문자형으로 출력한다.
15
16    d = '#';           ---- 문자형 변수 d에 '#'를 대입한다.
17    printf("%c의 ASCII 값은 %d 입니다 \n", d, d);  ---- 문자형 변수 d를 두 가지 형태로 출력한다.
18 }
```

실행 결과

```
P
A
#의 ASCII 값은 35 입니다
```

정답_ 1 printf("%c \n", b); 2 printf("%c \n", c);

5. 여러 글자가 모인 문자열과 배열

▪ 문자열의 기본 형식

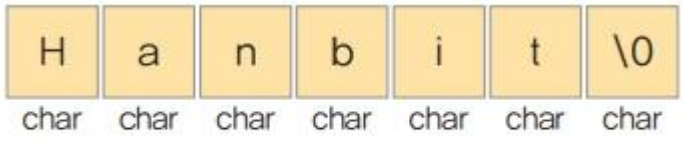

그림 3-24 문자열의 기본 구조

- 문자열은 문자형(char)의 집합이므로 "Hanbit"이라는 문자열을 저장하기 위해서는 문자 6개 가 나란히 있으면 됨
- [그림 3-24]에서는 문자가 6개가 아니라 7개인데 이를 주의해야 함

5. 여러 글자가 모인 문자열과 배열

▪ 문자열의 기본 형식

여기서 잠깐 올바른 문자 표현

- 문자는 반드시 ' '로 감싸야 하며 한 글자만 올 수 있으며 ❶~❸은 모두 틀린 표현
- char a;
- ❶ a = 'Ab'; ❷ a = "A"; ❸ a = "Ab";

- 문자열을 저장하는 변수는 문자형을 연속적으로 나열한 것을 의미하는 배열 형태가 되어야 함
- [그림 3-24]에서는 문자형 7개를 나열하여 배열로 생성
- 이것을 문법적으로는 다음과 같이 정의

```
char str[7];
```

- [ ] 안에는 필요한 문자의 개수를 쓰면 됨

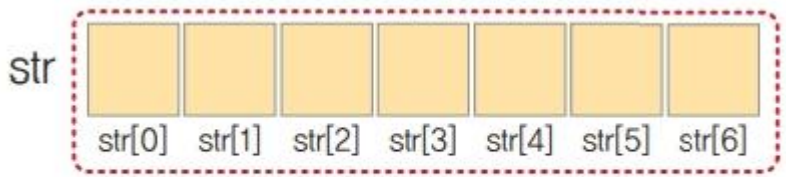

그림 3-25 배열로 표현한 문자열

5. 여러 글자가 모인 문자열과 배열

▪ 문자열의 기본 형식

- 전체 배열의 이름은 str이고 각각의 이름은 str[0]~str[6]
- 주의할 점은 배열이 0부터 시작하므로 마지막은 str[7]이 아닌 str[6]이라는 것
- 이때 [ ] 안에 들어가는 번호를 '첨자'라고 함
- 이번에는 str 배열에 "Basic"이라는 문자열을 대입
- 다른 데이터 형식은 대입 연산자(=) 를 사용해서 대입했으나 문자열은 특별히 strcpy( ) 함수를 사용해야 함
- 개념적으로는 대입 연산자와 비슷

```
strcpy(str, "Basic");
```

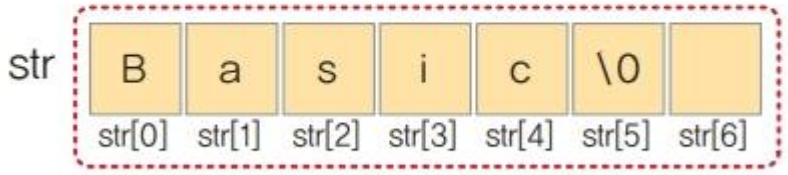

그림 3-26 str 배열에 문자열 대입

5. 여러 글자가 모인 문자열과 배열

▪ 문자열의 기본 형식

기본 3-14 문자열 형식 사용 예 1 — 3-14.c

```
01 #define _CRT_SECURE_NO_WARNINGS
02 #include <string.h>
03 #include <stdio.h>
04 void main( )
05 {
06    char str1[10];                  ----- 문자형 배열 str1과 str2를 선언한다.
07    char str2[10];
08    char str3[10] = "CookBook";     ----- 문자형 배열 str3을 선언함과 동시에 문자열을 대입한다.
09
10    strcpy(str1, "Basic-C");        ----- str1에 문자열을 대입한다.
11    strcpy(str2, str3);             ----- str3의 값을 str2에 복사한다.
12
13    printf("str1 ==> %s \n", str1); ----- 문자형 배열 str1, str2, str3을 출력한다.
14    printf("str2 ==> %s \n", str2);
15    printf("str3 ==> %s \n", str3);
16 }
```

실행 결과

```
str1 ==> Basic-C
str2 ==> CookBook
str3 ==> CookBook
```

5. 여러 글자가 모인 문자열과 배열

▪ 문자열의 기본 형식

- [기본 3-14]의 6행과 7행에서는 각각 문자형 배열 10개를 선언

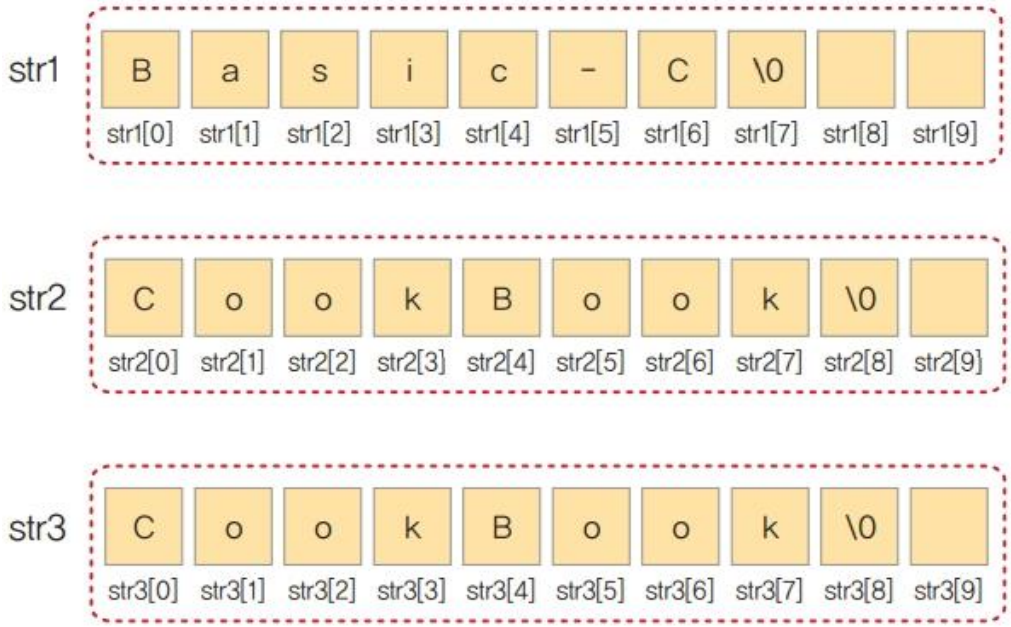

그림 3-27 [기본 3-14]의 문자열 선언 상태

5. 여러 글자가 모인 문자열과 배열

▪ 문자열과 배열의 뒷이야기

응용 3-15 문자열 형식 사용 예 2 — 3-15.c

```
01 #include <stdio.h>
02
03 void main( )
04 {
05    char str[10] = "0123456789";          ----- 열 자리의 str 배열에 글자 10개를 대입한다.
06
07    printf("str ==> %s \n", str);         ----- str의 내용을 출력한다.
08
09    str[0] = 'I';                         ----- str 배열에 글자 6개와 널 문자를 입력한다.
10    str[1] = 'T';
11    str[2] = 'C';
12    str[3] = 'o';
13    str[4] = 'o';
14    str[5] = 'k';
15    str[6] = '\0';
16
17    printf("str ==> %s \n", str);         ----- str의 내용을 출력한다.
18    printf("str[7] ==> %c \n", __1__ );   ----- str[7]의 한 글자를 출력한다.
19    printf("str[50] ==> %c \n", __2__ )   ----- str[50]의 한 글자를 출력한다.
20 }
```

정답_ 1 str[7] 2 str[50]

실행 결과

```
str ==> 0123456789徼徼徼?x?缸
str ==> ITCook
str[7] ==> 7
str[50] ==> ?
```

5. 여러 글자가 모인 문자열과 배열

▪ 문자열과 배열의 뒷이야기

- [그림 3-28]을 보면 str 배열에 널 문자가 없으므로 그 이후에(그림에서는 ???로 표시) 무엇이 들어 있는지 알 수 없지만 일단은 널 문자를 만날 때까지 계속 출력
- 널 문자는 문자열을 출력할 때 자동차의 브레이크와 같은 역할을 하므로, 이 경우는 브레이크 가 없어서 계속 달릴 수밖에 없는 상황

그림 3-28 [응용 3-15]의 str 배열 내용 1

5. 여러 글자가 모인 문자열과 배열

▪ 문자열과 배열의 뒷이야기

- 9~15행은 str 배열에 strcpy( ) 함수를 이용하여 값을 대입하지 않고 한 글자씩 직접 대입한 것
- 즉 [그림 3-29]와 같이 첨자(str[n]과 같은 형식)를 이용하면 한 자리씩 접근할 수 있음

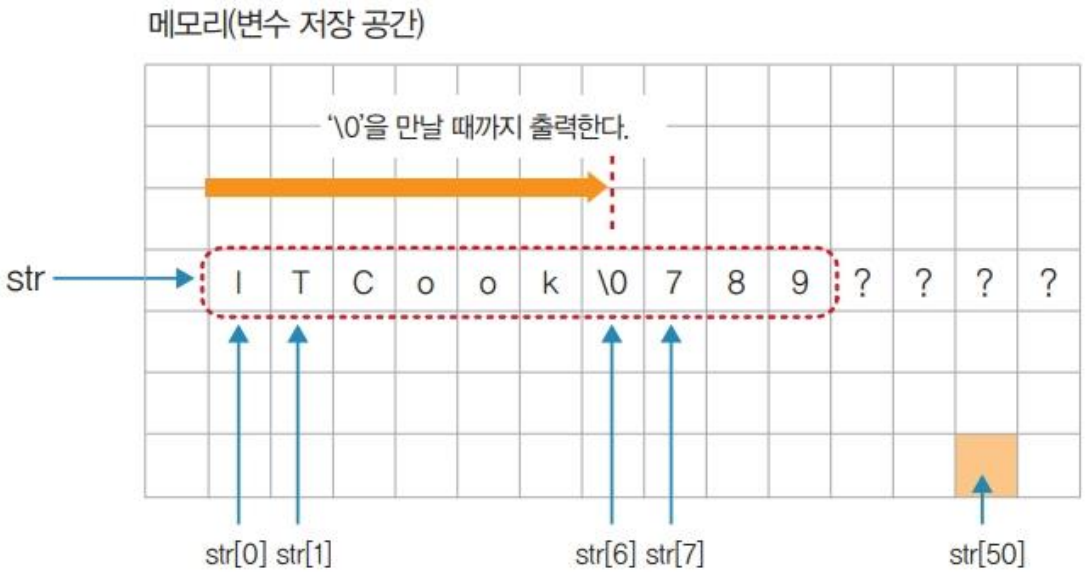

그림 3-29 [응용 3-15]의 str 배열 내용 2

*

예제 모음

[예제모음 04] 정수형을 출력하는 프로그램

예제 설명 정수를 하나 입력받아 10진수, 16진수, 8진수로 출력하는 프로그램이다.

실행 결과

```
정수를 입력하세요 ==> 9999
10진수 ==> 9999
16진수 ==> 270F
8진수 ==> 23417
```

[예제모음 04] 정수형을 출력하는 프로그램

```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    int data;                          ----- 정수형 변수를 선언한다.
06
07    printf("정수를 입력하세요 ==> ");
08    scanf("%d", &data);                ----- 키보드로 정수를 입력받는다.
09
10    printf("10진수 ==> %d \n", data);  ----- 10진수(%d), 16진수(%X), 8진수(%o)를 출력한다.
11    printf("16진수 ==> %X \n", data);
12    printf("8진수 ==> %o \n", data);
13 }
```

[예제모음 05] 입력하는 정수의 진수 결정

예제 설명 10진수, 16진수, 8진수 중 어떤 진수의 값을 입력받을지 결정하고, 입력받은 수를 10진수, 16진수, 8진수로 출력하는 프로그램이다.

실행 결과

```
입력진수 결정 <1>10 <2>16 <3>8 : 2
값 입력 : FF
10진수 ==> 255
16진수 ==> FF
8진수 ==> 377
```

[예제모음 05] 입력하는 정수의 진수 결정

```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    int type, data;
06
07    printf("입력진수 결정 <1>10 <2>16 <3>8 : ");
08    scanf("%d", &type);          ----- 키보드로 1~3 중 하나를 입력받는다.
09
10    printf("값 입력 : ");
11
12    if(type == 1)                ----- 입력값이 1이면 10진수를 입력받는다.
13    {   scanf("%d", &data);   }
14
15    if(type == 2)                ----- 입력값이 2이면 16진수를 입력받는다.
16    {   scanf("%x", &data);   }
17
18    if(type == 3)                ----- 입력값이 3이면 8진수를 입력받는다.
19    {   scanf("%o", &data);   }
20
21    printf("10진수 ==> %d \n", data);   ----- 입력받은 data 값을 10진수, 16진수,
22    printf("16진수 ==> %X \n", data);       8진수로 변환하여 출력한다.
23    printf("8진수 ==> %o \n", data);
24 }
```

[예제모음 06] 데이터 형의 크기 확인

예제 설명 sizeof( ) 함수를 사용해서 각 데이터형의 크기를 확인하는 프로그램이다.

실행 결과

```
int 형의 크기                ==> 4
unsigned int 형의 크기         ==> 4
short 형의 크기              ==> 2
unsigned short 형의 크기       ==> 2
long int 형의 크기           ==> 4
unsigned long int 형의 크기    ==> 4
float 형의 크기              ==> 4
double 형의 크기             ==> 8
long double 형의 크기          ==> 8
char 형의 크기               ==> 1
unsigned char 형의 크기        ==> 1
```

[예제모음 06] 데이터 형의 크기 확인

```
01 #include <stdio.h>
02
03 void main( )
04 {
05   printf("int 형의 크기\t\t\t ==> %d\n", sizeof(int));
06   printf("unsigned int 형의 크기\t\t ==> %d\n", sizeof(unsigned int));
07   printf("short 형의 크기\t\t\t ==> %d\n", sizeof(short));
08   printf("unsigned short 형의 크기\t ==> %d\n", sizeof(unsigned short));
09   printf("long int 형의 크기\t\t ==> %d\n", sizeof(long int));
10   printf("unsigned long int 형의 크기\t ==>
       %d\n", sizeof(unsigned long int));
11   printf("float 형의 크기\t\t\t ==> %d\n", sizeof(float));
12   printf("double 형의 크기\t\t ==> %d\n", sizeof(double));
13   printf("long double 형의 크기\t\t ==> %d\n", sizeof(long double));
14   printf("char 형의 크기\t\t\t ==> %d\n", sizeof(char));
15   printf("unsigned char 형의 크기\t\t ==> %d\n", sizeof(unsigned char));
16 }
```

sizeof( ) 함수로 각 데이터형의 크기(바이트 수)를 출력한다. 이때 컴파일러에 따라서 long double 형은 16바이트 크기일 수도 있다.

[예제모음 07] 입력된 문자열을 거꾸로 출력

예제 설명 열 글자 미만의 문자열을 입력받고, 입력받은 문자열을 반대 순서로 출력하는 프로그램이다(아직 배우지 않은 내용이 나오지만 나중을 위해 미리 살펴보자).

실행 결과

```
문자열을 입력 ==> ABCDEFG
GFEDCBA
```

[예제모음 07] 입력된 문자열을 거꾸로 출력

```
01 #define  _CRT_SECURE_NO_WARNINGS
02 #include <stdio.h>
03 void main( )
04 {
05    char str[10]=" ";          ---- 문자열을 입력받을 str 배열을 준비한다.
06    int i;                     ---- 첨자를 준비한다.
07
08    printf("문자열을 입력 ==> ")
09    scanf("%s", str);          ---- 문자열을 입력받는다.
10
11    for(i = sizeof(str) - 1; i >= 0; i--)
12    {                          ---- str 배열에 들어 있는 문자열을
13       printf("%c", str[i]);        맨 뒤의 str[9]부터 str[0]까지
14    }                               출력한다. 즉 입력한 순서의 반대로
15    printf("\n");                   출력되는 것이다.
16 }
```

감사합니다!