



[2주차 2강]

입출력



학습 내용

3.1 printf()를 이용한 출력

3.2 scanf()를 이용한 입력



학습 목표

- 3.1 화면에 값을 출력하기 위해 `printf()` 함수의 활용법을 익힌다.
- 3.2 사용자로부터 값을 입력 받기 위한 `scanf()` 함수의 활용법을 익힌다.





3.1 printf()를 이용한 출력

3.2 scanf()를 이용한 입력



3.1 printf()를 이용한 출력



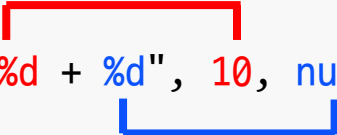
기본 사용법

- 출력할 내용

```
printf("출력하고 싶은 내용");
```

- 서식 지정자를 이용한 값 출력

```
printf("%d + %d", 10, num);
```



3.1 printf()를 이용한 출력



서식 지정자와 출력할 값 개수 불일치

- 서식 지정자가 더 많은 경우 : 남은 서식 지정자는 쓰레기 값

```
printf("%d + %d = %d", 3, 4);
```

- 출력할 값이 더 많은 경우 : 남은 값 사용되지 않음

```
printf("%d + %d", 3, 4, 7);
```

3.1 printf()를 이용한 출력



서식 지정자 종류

- %d, %f, %c 이외에 다양한 서식 지정자

실행결과

```
printf("%d\n", -10);
```

-10 ⇒ -10 출력

```
printf("%o\n ", 26);
```

32 ⇒ $26_{(10)} = 32_{(8)}$

```
printf("%x\n", 26);
```

1a ⇒ $26_{(10)} = 1a_{(16)}$

```
printf("%f\n ", 123.45);
```

123.450000 ⇒ 123.45

```
printf("%e\n", 123.45);
```

1.234500e+02 ⇒ 1.2345×10^2

3.1 printf()를 이용한 출력



서식 지정자 종류

- printf의 주요 서식 지정자 정리 (* 이 외의 서식 지정자는 교재 참고)

분류	서식 지정자	대상 자료형	출력 형태
정수형	%d	int	10진수
	%u	unsigned int	10진수
	%o		8진수
	%x, %X		16진수
부동소수형	%f	float, double	고정 소수점 표기
문자형	%c	char	문자 하나

3.1 printf()를 이용한 출력



주의!!

- 출력 형태를 결정하는 것은 자료형이 아니라 서식 지정자
✓ 100을 '%d'로 출력, '%c'로 출력

```
printf("정수로 출력: %d\n", 100);
```

```
printf("문자로 출력: %c\n", 100);
```

실행결과

```
정수로 출력: 100
```

```
문자로 출력: d
```

3.1 printf()를 이용한 출력



서식 지정자 확장

출력 폭 지정

```
printf("%5d\n", 25 );
```

			2	5
--	--	--	---	---

```
printf("%9f\n", 2.5 );
```

	2	.	5	0	0	0	0	0
--	---	---	---	---	---	---	---	---

정밀도 지정(부동소수)

```
printf("%.3f\n",2.5);
```

2	.	5	0	0
---	---	---	---	---

```
printf("%.1f\n",2.56);
```

2	.	6
---	---	---

```
printf("%.2f\n",2.5);
```

		2	.	5	0
--	--	---	---	---	---



왜 숫자 6이
출력되는지 생각해보자

3.1 printf()를 이용한 출력



서식 지정자 확장

- 플래그
 - ✓ 정렬방식, 부호 출력 방식 진법 표시 방식 등을 조정하기 위해 사용
- 0 플래그

```
printf("%05d\n", 25 );
```

0

0

0

2

5

- ✓ 다른 플래그에 대해서는 교재 참고

실습하기



[예제 3.1]

125와 12.56을 아래와 같이 출력되도록 각각의 문장을 작성하시오.



실행화면

1	2	5						
			1	2	5			
0	0	1	2	5				
1	2	.	5	6	0	0	0	0
		1	2	.	6			
	1	2	.	5	6	0		
0	0	1	2	.	5	6		

```
printf("%d\n", 125);
```

[빈 칸]

[빈 칸]

```
printf("%f\n", 12.56);
```

[빈 칸]

[빈 칸]

[빈 칸]



3.1 printf()를 이용한 출력

3.2 scanf()를 이용한 입력



3.2 scanf()를 이용한 입력



기본 사용법

- 서식 지정자를 쓰고, 뒤에 값을 저장할 변수 이름 명시
- 반드시 & 기호 붙이기(특별한 언급이 없으면)

```
scanf("%d", &a);
```

- 자료형에 따라 사용되는 서식 지정자가 다름
 - ✓ 서식 지정자와 자료형이 일치하지 않으면, 오류 발생

3.2 scanf()를 이용한 입력

주요 서식 지정자(이 외의 서식 지정자는 교재 참고)

분류	서식 지정자	대상 자료형
정수형	%d	int
부동소수형	%f	float
	%lf	double
문자형	%c	char

정리

정수	<ul style="list-style-type: none"> 기본적으로 int 형으로 선언하고, '%d'로 출력 및 입력
부동소수	<ul style="list-style-type: none"> 기본적으로 double 형으로 선언하고, '%f'로 출력(%lf 도 가능), '%lf'로 입력

3.2 scanf()를 이용한 입력

입력 형태를 결정하는 것은 자료형이 아니라 서식 지정자

- 3을 입력했을 때, 3은 정수인가? 문자인가?

실행결과

```
int a = 0;

printf("문자 입력: ");
scanf("%c", &a); ⇒ 문자로 입력
printf("a: %c %d\n", a, a);

printf("정수 입력: ");
scanf("%d", &a); ⇒ 정수로 입력
printf("a: %c %d\n", a, a);
```

문자 입력: 3

a: 3 51

정수 입력: 3

a: 3 3

3.2 scanf()를 이용한 입력



여러 값 입력 받기

- 예) 정수와 부동소수 입력 받기 : 값 구분 위해 보통 공백 사용

```
int a;  
double b;  
  
scanf("%d%lf", &a, &b);  
printf("입력 값: %d %f\n", a, b);
```

공백입력

실행결과

5 3.1[Enter]

입력 값: 5 3.100000

[Enter]는 엔터 키
입력을 의미

3.2 scanf()를 이용한 입력



여러 문자 입력 시 주의할 점

- 공백도 하나의 문자로 간주되어 입력 값으로 처리되므로 공백 없이 입력

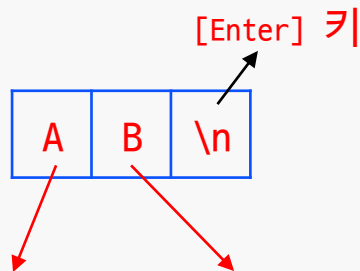
```
char c1, c2;
printf("문자 두 개 입력: ");
scanf("%c%c", &c1, &c2 );
printf("문자 출력: [%c][%c]\n", c1, c2 );
```

실행결과

문자 두 개 입력: AB[Enter]

문자 출력: [A][B]

공백 없이
연달아 입력



c1에 저장 c2에 저장

3.2 scanf()를 이용한 입력



여러 문자 입력 시 주의할 점

- 탭, 개행 문자도 동일하게 문자 입력으로 처리

```
char c1, c2;

printf("c1 입력: ");
scanf("%c", &c1 );           ⇒ 첫 번째 문자 입력
printf("c2 입력: ");
scanf("%c", &c2 );           ⇒ 두 번째 문자 입력

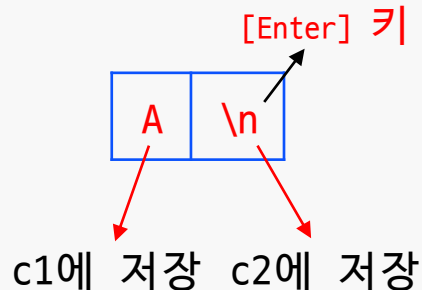
printf("문자 출력: [%c][%c]\n", c1, c2 );
```

실행결과

c1 입력: A[Enter]

C2 입력: 문자 출력: [A][
]

개행 문자(Enter키) 출력



3.2 scanf()를 이용한 입력



여러 문자 입력 시 주의할 점

- 원하는 형태로 입력받기 위한 해결책(트릭)
 - ✓: [Enter] 키로 입력되는 **개행 문자를 임시 변수에 저장해 없애기**

```
char c1, c2, tmp;

printf("c1 입력: ");
scanf("%c%c", &c1, &tmp );    ⇒ 첫 번째 문자 입력
printf("c2 입력: ");
scanf("%c%c", &c2, &tmp );    ⇒ 두 번째 문자 입력

printf("문자 출력: [%c][%c]\n", c1, c2 );
```

실행결과

c1 입력: A[Enter]

c2 입력: B[Enter]

문자 출력: [A][B]

→ **개행 문자는 tmp에 저장**
(이런 목적으로는 나중에 배울
getchar() 함수를 더 자주 사용)

3.2 scanf()를 이용한 입력



문자와 정수(또는 부동소수)가 혼합되어 입력 받는 경우

- 정수 입력에서는 공백, 탭, 개행 문자 무시됨
- 문자 입력에서는 공백, 탭, 개행 문자가 하나의 문자로 처리됨

```
int n;          char c1, c2;

printf("문자 입력: ");
scanf("%c", &c1 );           ⇒ c1에 문자 저장
printf("정수 입력: ");
scanf("%d", &n );            ⇒ n에 정수 저장
printf("문자 입력: ");
scanf("%c", &c2 );           ⇒ c2에 문자 저장

printf("출력: [%c][%d][%c]\n", c1, n, c2 );
```

결과를 예상해보고 직접 실행해 확인해 보자.

실행결과

문자 입력: A[Enter]

정수 입력: 25[Enter]

...(이후 결과는??)

3.2 scanf()를 이용한 입력



입력 구분자 지정하기

- 입력 구분자로 하이픈(-)을 사용한 예

```
int a=0, b=0, c=0;  
  
printf("정수 입력: ");  
  
scanf("%d-%d-%d", &a, &b, &c);  
printf("%d %d %d\n", a, b, c);
```

실행결과(입력 형식을 맞춘 예)

정수 입력: 20-10-30[Enter]

20 10 30

실행결과(입력 형식을 맞추지 않은 예)

정수 입력: 20 10 30[Enter]

20 0 0

3.2 scanf()를 이용한 입력



[예제 3.2]

- 두 개의 문자 'A'와 'B'를 공백을 구분하여 입력 받고, 정상적으로 입력되었는 지를 값을 출력하여 확인해보자.

```
char c1, c2;  
  
printf("문자 입력: ");  
[ 빈 칸 ]  
printf("문자 출력: [%c][%c]\n", c1, c2 );
```

실행결과

문자 입력: AB[Enter]

문자 출력: [A][B]

공백 입력

3.2 scanf()를 이용한 입력



scanf 사용 시 주의 사항

- 특별한 목적이 없는 한 "..." 안에 서식 지정자 이외의 내용은 넣지 말자

```
scanf("%d\n", &a );
```

→ 개행 문자를 넣지 않도록 하자.

```
scanf("정수입력: %d", &a );
```

→ 안내 문구를 넣지 않도록 하자.

학습 정리

- 정수는 기본적으로 **int** 형으로 선언하고 출력과 입력에 **서식지정자로 %d**를 사용함
- 부동소수는 기본적으로 **double** 형으로 선언하고 **출력** 시에는 '**%f**', **입력** 시에는 '**%lf**'를 사용함
- **scanf()** 함수에서는 값을 저장할 변수 앞에 **기호 & (주소연산자)**를 붙여야 함 (기호 &를 안 붙여도 되는 경우는 나중에 학습한다.)
- 키보드로 입력하는 정보를 어떤 자료형으로 해석할지는 결정하는 것은 **서식 지정자**임
- **문자 입력 시** [Space] 키나 [Enter] 키도 하나의 문자로 간주되고, 문자와 정수를 혼합하여 입력 받는 경우, 공백이나 개행 문자에 대한 처리에 주의해야 함
- **scanf()** 함수에서 서식 지정자 이외의 문자는 **입력 구분자** 역할을 하므로, 특별한 경우가 아니면 서식 지정자만 명시해야 함