Formatted Input/Output

adopted from KNK C Programming : A Modern Approach

The `printf` Function (1/3)

- `printf` 함수는 출력될 문자열과 해당 문자열에 포함되어야 할 값들로 구성되어 있음

  `printf(`*format_string*`, `*expr*`1, `*expr*`2, …);`

- 출력될 문자열은 일반 글자들과 %로 시작되는 형식지정자가 포함될 수 있음

- 형식 지정자는 출력될 값이 표현될 위치를 나타냄
  - `%d` 는 `int` 형 값에 쓰임
  - `%f` 는 `float` 형 값에 쓰임

The `printf` Function (2/3)

- 일반 문자는 기록된 데로 표현되고 형식지정자는 뒤 따라오는 변수/값을 표혀함
- Example:

```
int i, j;
float x, y;

i = 10;
j = 20;
x = 43.2892f;
y = 5527.0f;

printf("i = %d, j = %d, x = %f, y = %f\n", i, j, x, y);
```

Result:

The `printf` Function (3/3)

- 컴파일러는 몇개의 형식지정자가 사용되었는지 검사하지 않아도 됨.
  하지만, 형식지정자의 개수와 변수/값의 수는 같아야 함

```
printf("%d %d\n", i);    /*** WRONG ***/
printf("%d\n", i, j);    /*** WRONG ***/
```

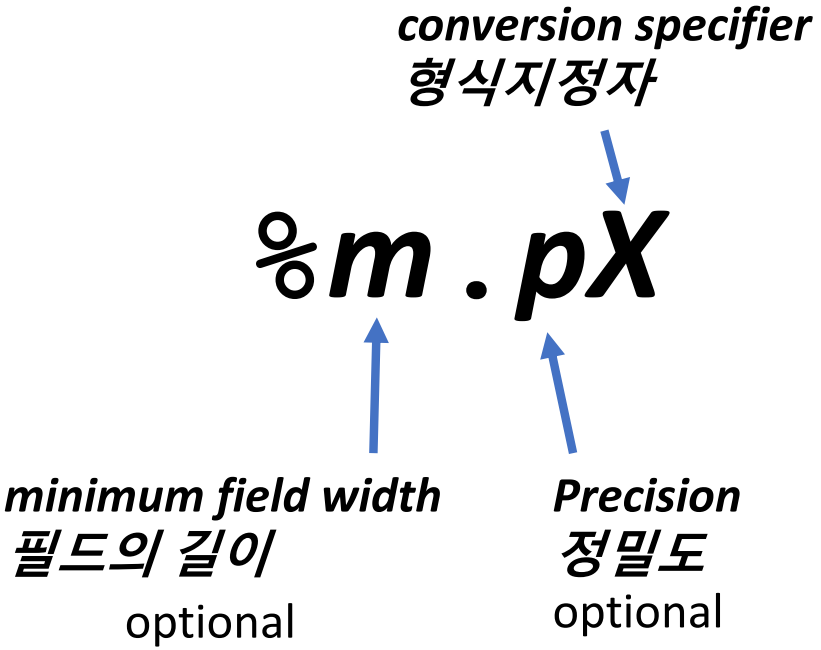
- 컴파일러는 형식지정자가 제대로 쓰였는지 검사하지 않아도 됨
  - 형식이 맞지 않으면 의미 없는 결과를 출력하게 됨:

```
int i;

float x;

printf("%f %d\n", i, x);   /*** WRONG ***/
```

Conversion Specifications형식 지정자 (1/2)



	12345.6789
%5.3f	12345.678
%8d	12345
%-8d	12345

- %d – Integer 정수
- %e – Exponential format 지수
- %f – Fixed decimal 소수점
- %g – Either exponential format or fixed decimal format
  지수형이나 소수점 형

Conversion Specifications (1/2)

Format specifier	Description	Supported data types
%c	Character	char unsigned char
%d	Signed Integer	short unsigned short int long
%e or %E	Scientific notation of float values	float double
%f	Floating point	float
%g or %G	Similar as %e or %E	float double
%hi	Signed Integer(Short)	short
%hu	Unsigned Integer(Short)	unsigned short
%i	Signed Integer	short unsigned short int long
%l or %ld or %li	Signed Integer	long
%lf	Floating point	double
%Lf	Floating point	long double
%lu	Unsigned integer	unsigned int unsigned long

Format specifier	Description	Supported data types
%lli, %lld	Signed Integer	long long
%llu	Unsigned Integer	unsigned long long
%o	Octal representation of Integer.	short unsigned short int unsigned int long
%p	Address of pointer to void void *	void *
%s	String	char *
%u	Unsigned Integer	unsigned int unsigned long
%x or %X	Hexadecimal representation of Unsigned Integer	short unsigned short int unsigned int long
%n	Prints nothing	
%%	Prints % character	

Escape Sequences 특수 문자 (1/2)

- `\n` 와 같은 서식을 ***escape sequence 특수문자라 함.***

- 특수 문자는 제어용 출력이 안되는 문자와 특별한 의미를 갖는 “와 같은 문자들로 구성되어 있음

- 특수 문자의 일부:

Alert (bell, 종소리)	`\a`
Backspace 백스페이스	`\b`
New line 줄 바꿈	`\n`
Horizontal tab 탭 문자	`\t`

Escape Sequences (2/2)

- 문자열에는 특수문자가 몇이든 포함될 수 있음:

```
printf("Item\tUnit\tPurchase\n\tPrice\tDate\n");
```

```
Item   Unit   Purchase
       Price  Date
```

- 주로 사용되는 특수문자는 \" 로서 " 문자를 출력함:

```
printf("\"Hello!\""); /* prints "Hello!" */
```

- \ 문자를 쓰려면, \ 문자를 두번 연속으로 쓰면 됨:

```
printf("\\");          /* prints one \ character */
```

The `scanf` Function

- `scanf` 는 특정 형식으로 입력을 읽음

  `scanf (`*format_string*`, &var1, &var2, …);`

- `scanf` 의 문자 형식은 **일반 문자**와 **형식지정자**
- `scanf` 의 형식 변환은 `printf`와 동일함.

- 많은 경우 `scanf` 의 문자 형식은 형식지정자만 포함하고 있음:

```
int i, j;
float x, y;
scanf("%d%d%f%f", &i, &j, &x, &y);
```

- 예제 입력: `1 -20 .3 -4.0e3`

  `scanf` 는 1, –20, 0.3,–4000.0 을 `i, j, x, y`에 저장함

How `scanf` Works (1/4)

- `scanf` 는 입력된 글자들을 형식지정자에 매치를 시키는 일을 함
- 각 형식지정자에 대해 `scanf` 는 공백문자는 제외하고 입력 값을 해당 형으로 변환함
- `scanf` 가 입력된 값을 읽어 들인 후 형식에 맞지 않는 글자가 나오면 멈춤
- 정보를 제대로 읽었으면 `scanf`는 다음 문자 형식을 처리함
  - 더 읽을 것이 없으면 `scanf` 는 즉시 리턴함

How `scanf` Works (2/4)

- 숫자를 찾는 동안 공백 문자는 무시함
  - 스페이스, 탭, 줄바꿈 등
- `scanf` 호출로 4개의 수를 읽는 예

```
scanf("%d%d%f%f", &i, &j, &x, &y);
```

- 이 경우 입력이 여러 줄에 걸쳐 입력될 수 있음

```
  1
-20    .3
   -4.0e3
```

```
••1¤-20•••.3¤•••-4.0e3¤
ssrsrrrsssrrsssssrrrrrr
```

(`s` = 건너뜀; `r` = 읽기)

- `scanf` 는 마지막 줄 바꿈 기호를 읽지는 않고 "엿보기"만 한다

How `scanf` Works (3/4)

- 정수를 읽으려고 하면 `scanf` 는 먼저 숫자와 더하기 또는 빼기 기호를 찾음. 그리고 숫자가 아닌 것이 나올 때까지 읽음

- 소수점을 읽으려고 하면 다음의 순서대로 정보를 찾음
  - 덧셈, 뺄셈 기호 (optional), 그리고
  - 숫자 (소수점을 포함하는), 그리고
  - 지수 (optional). 지수는 문자 `e` (또는 `E`)를 쓰고, 양수음수 부호와 하나 또는 그 이상의 자리수로 구성.
- `scanf` 에서 `%e`, `%f`, `%g` 는 서로 교환이 됨.

- 만약 `scanf` 가 이번 읽기 시도에서 포함이 불가능한 문자를 만나게 되면 그 문자의 읽기를 취소함

How `scanf` Works (4/4)

- 예제 입력:

  `1-20.3-4.0e3¤`

- `scanf` 의 호출은 앞의 예에서와 같음:

```
scanf("%d%d%f%f", &i, &j, &x, &y);
```

- 새로운 입력을 `scanf` 가 어떻게 처리하는 지 보자:
  - `%d` : 1을 `i`에 저장하고 `-` 문자는 되돌려 놓음
  - `%d` : –20 을 `j`에 저장하고 `.` 문자는 되돌려 놓음
  - `%f` : 0.3 을 `x` 에 저장하고 `-` 문자는 되돌려 놓음
  - `%f` : –4.0 × 103 을 `y` 에 저장하고 줄바꿈 문자는 되돌려 놓음

형식 문자열의 일반 문자

- 하나 또는 그 이상의 공백 문자를 형식 문자열에서 만나면 `scanf` 는 공백문자가 아닌 문자를 만날 때까지 공백 문자를 계속 읽음

- 공백문자가 아닌 문자를 만나면 `scanf` 입력받아야 하는 형과 같은지 비교함
  - 같으면, `scanf` 다음 번 처리할 형식으로 이동함
  - 다르면, `scanf` 다른 문자를 복원하고 종료함

- Examples:
  - 형식 문자열이 "`%d/%d`" 이고 입력이 `•5/•96` 이면, `scanf` 성공.
  - 입력이 `•5•/•96` 이면, `scanf` 실패, `/` 문자가 형식문자열에 지정한 값과 다르기 때문
- 공백을 허용하려면 "`%d /%d`" 이렇게 써야 함.

printf 와 scanf 주의점 (1/2)

- scanf 과 printf 의 호출이 유사해 보이지만, 매우 다른 함수임

- 흔한 실수: & 를 printf의 인자에 쓰는 경우
```
printf("%d %d\n", &i, &j);  /*** WRONG ***/
```

- scanf 의 형식 문자열이 printf 의 형식 문자열과 같아야 한다고 착각하면 안됨
- scanf의 호출 예를 보자:
```
scanf("%d, %d", &i, &j);
```
  - scanf 는 먼저 입력으로 정수를 기대하며, 그 값을 i에 저장.
  - scanf 은 그리고 쉼표가 입력되기를 기대함
  - 입력이 쉼표가 아니라 공백이면 scanf 는 j를 위한 값을 읽지 않고 종료함.

printf 와 scanf 주의점(2/2)

- scanf 의 형식 문자열의 끝에 줄바꿈 기호를 넣는 것은 좋은 생각이 아님
- Scanf는 줄바꿈 기호를 공백과 같이 처리하기 때문에 다음 칸으로 이동과 같은 의미로 사용됨
- "%d\n" 이라고 쓰면, scanf 는 공백 문자를 무시하고 정수를 읽은 뒤 다시 공백문자가 아닌 글자가 들어오기를 기다리게 됨
- 이런 경우 프로그램이 멈춘 것처럼 보이게 됨