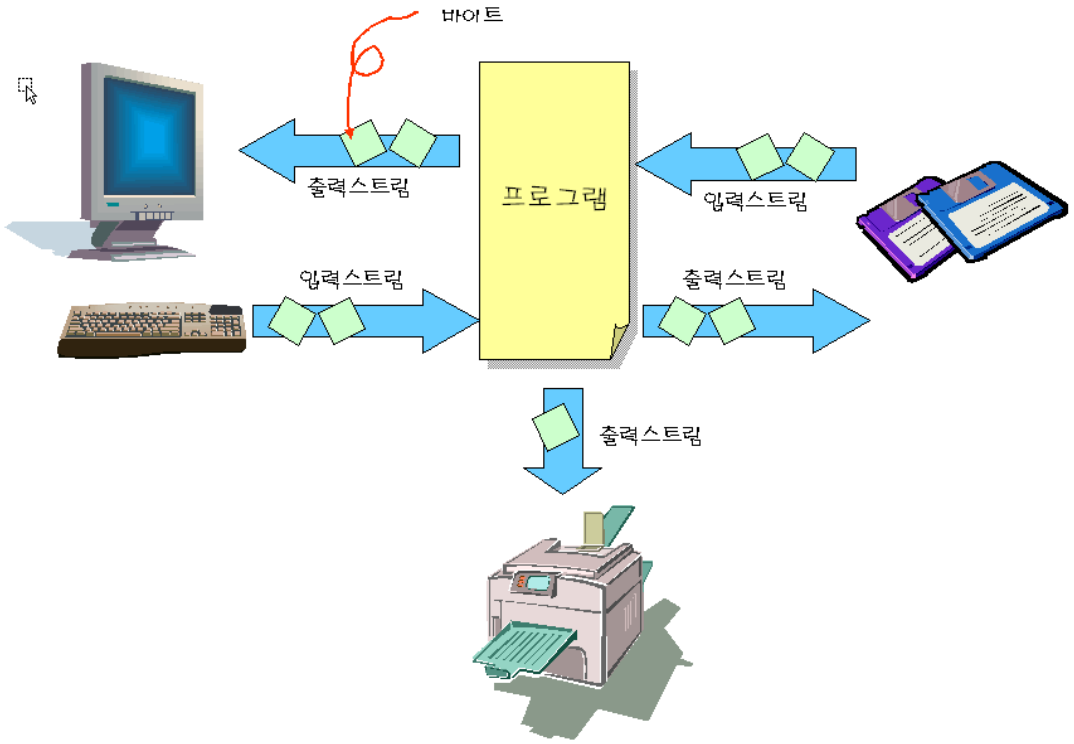
제14장
입출력과 라이브러리 함수

Stream의 개념

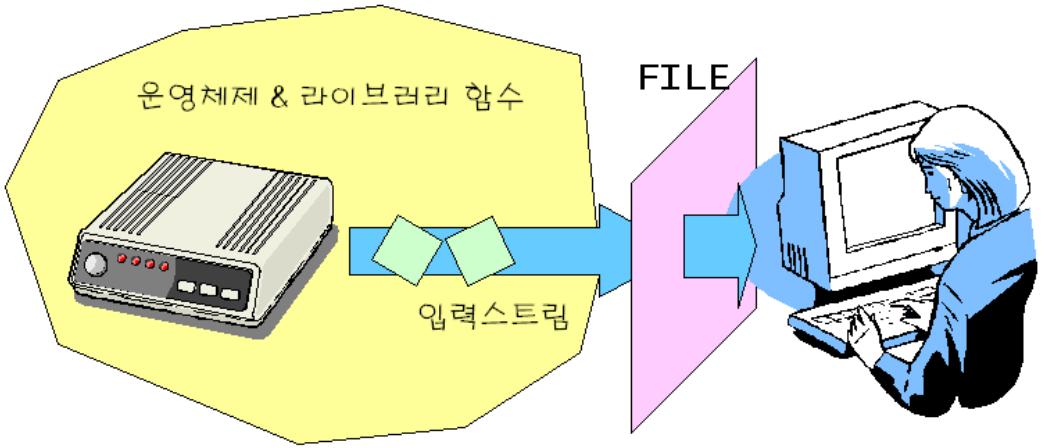
- 스트림(stream): 모든 입, 출력 및 장치들을 바이트(byte)들의 흐름으로 생각하는 것



스트림과 파일

- 스트림은 구체적으로 FILE 구조체를 통하여 구현
- FILE은 stdio.h에 정의되어 있다.

```
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
};
typedef struct _iobuf FILE;
```



표준 입출력 스트림

이름	스트림	연결 장치
stdin	표준 입력 스트림	키보드
stdout	표준 출력 스트림	모니터의 화면
stderr	표준 에러 스트림	모니터의 화면

- 표준 입출력 스트림(standard input/output stream): 필수적인 몇 개의 스트림
- 프로그램 실행 시에 자동으로 만들어지고 프로그램 종료 시에 자동으로 없어진다.
- 스트림의 최대 개수는 512개
- 3개의 표준 입출력 스트림이 첫 부분을 차지
- stdin은 표준 입력 스트림
- stdout은 표준 출력 스트림
- stderr은 에러를 따로 출력하기 위하여 만들어진 스트림

입출력 함수의 분류

스트림 형식	표준 스트림	일반 스트림	설명
형식이 없는 입출력 (문자 형태)	getchar()	fgetc(FILE *f,...)	문자 입력 함수
	putchar()	fputc(FILE *f,...)	문자 출력 함수
	gets()	fgets(FILE *f,...)	문자열 입력 함수
	puts()	fputs(FILE *f,...)	문자열 출력 함수
형식이 있는 입출력 (정수, 실수,..)	printf()	fprintf(FILE *f,...)	형식화된 출력 함수
	scanf()	fscanf(FILE *f,...)	형식화된 입력 함수

- 사용하는 스트림에 따른 분류
  - 표준 입출력 스트림을 사용하여 입출력을 하는 함수
  - 스트림을 구체적으로 명시해 주어야 하는 입출력 함수
- 데이터의 형식에 따른 분류
  - getchar()나 putchar()처럼 *문자* 형태의 데이터를 받아들이는 입출력
  - printf()나 scanf()처럼 구체적인 형식을 지정할 수 있는 입출력

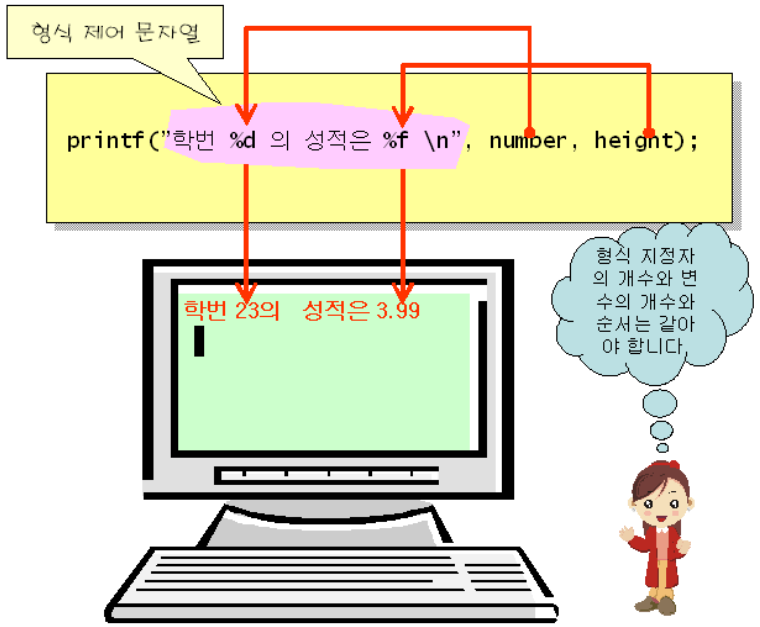
printf()를 이용한 출력

```
int printf(char *format, ...);
```

- 형식 제어 문자열의 구조

```
%[플래그] [필드폭] [.정밀도] [{h | l | L}] 형식
```

- % 기호
  - 형식 제어 문자열의 시작
- 플래그(flag)
  - 출력의 정렬과 부호 출력, 공백 문자 출력, 소수점, 8진수와 16진수 접두사 출력
- 필드폭(width)과 정밀도(precision)
  - 데이터가 출력되는 필드의 크기
  - 정밀도는 소수점 이하 자릿수의 개수가 된다.



필드폭

- 필드폭(field width)
  - 데이터가 출력되는 필드의 크기

형식 지정자	의미	
%6d	폭은 6이며 우측정렬하여 출력	␣ ␣ ␣ 1 2 3
%-6d	폭은 6이며 좌측정렬하여 출력	1 2 3 ␣ ␣ ␣
%+6d	폭은 6이며 우측정렬, 부호를 함께 출력	␣ ␣ + 1 2 3

```
#include <stdio.h>

int main(void)
{
        printf("%6d        %6d\n", 1, -1);
        printf("%6d        %6d\n", 12, -12);
        printf("%6d        %6d\n", 123, -123);
        printf("%6d        %6d\n", 1234, -1234);
        printf("%6d        %6d\n", 12345, -12345);
        printf("%6d        %6d\n", 123456, -123456);
        printf("%6d        %6d\n", 1234567, -1234567);
}
```

```
1 -1
12 -12
123 -123
1234 -1234
12345 -12345
123456 -123456
1234567 -1234567
```

정밀도

- 정밀도(precision)
  - 정수인 경우, 출력할 숫자의 개수
  - 실수인 경우, 소수점 이하의 자릿수의 개수

형식 지정자	의미	
%6.4d	6자리 중에서 4자리로 출력	□ □ 0 1 2 3
%6.2f	전체폭은 6, 소수점 이하 자리 2, 우측정렬	□ □ 1 . 2 3

```
#include <stdio.h>

int main(void)
{
        printf("%.5d          %.8d\n", 123, 123);

        printf("%.6f\n", 0.12345678);
        printf("%.6e\n", 0.12345678);
        printf("%.6g\n", 0.12345678);
        printf("%.6s\n", "Hello World !");
}
```

```
00123 00000123
0.123457
1.234568e-001
0.123457
Hello
```

플래그

기호	의미	기본값
-	출력 필드에서 출력값을 왼쪽 정렬한다.	오른쪽 정렬된다.
+	결과 값을 출력할 때 항상 +와 -의 부호를 붙인다.	음수일 때만 - 부호를 붙인다.
0	출력값 앞에 공백 문자 대신에 0으로 채운다. -와 0이 동시에 있으면 0은 무시된다. 만약 정수 출력의 경우, 정밀도가 지정되면 역시 0은 무시된다(예를 들어서 %08.5).	채우지 않는다.
blank(' ')	출력값 앞에 양수나 영인 경우에는 부호대신 공백을 출력한다. 음수일 때는 -가 붙여진다. + 플래그가 있으면 무시된다.	공백을 출력하지 않는다.
#	8진수 출력 시에는 출력값 앞에 0을 붙이고 16진수 출력 시에는 0x를 붙인다.	붙이지 않는다.

예제

```
#include <stdio.h>

int main(void)
{
		printf("-------------------------------------------------\n");
		printf("| 형식지정자  |   36인경우|     -36인경우|\n");
		printf("-------------------------------------------------\n");
		printf("|%%15d        |%15d |%15d |\n", 36, -36);
		printf("|%%-15d       |%-15d |%-15d |\n", 36, -36);
		printf("|%%+15d       |%+15d |%+15d |\n", 36, -36);
		printf("|%%015d       |%015d |%015d |\n", 36, -36);
		printf("|%% 15d       |% 15d |% 15d |\n", 36, -36);
		printf("|%%- 15d      |%- 15d |%- 15d |\n", 36, -36);
		printf("|%%#x         |%#x             |%#x |\n", 36, -36);
		printf("-------------------------------------------------\n");
}
```

```
-----------------------------------------------
| 형식지정자 | 36인 경우 | -36인 경우 |
-----------------------------------------------
|%15d | 36 | -36 |
|%-15d |36 |-36 |
|%+15d | +36 | -36 |
|%015d |000000000000036 |-00000000000036 |
|% 15d | 36 | -36 |
|%- 15d | 36 |-36 |
|%#x |0x24 |0xffffffdc |
-----------------------------------------------
```

정수출력

형식 지정자	설명	출력예
%d	부호있는 10진수 형식으로 출력	255
%i	부호있는 10진수 형식으로 출력	255
%u	부호없는 10진수 형식으로 출력	255
%o	부호없는 8진수 형식으로 출력	377
%x	부호없는 16진수 형식으로 출력, 소문자로 표기	fe
%X	부호없는 16진수 형식으로 출력, 대문자로 표기	FE

```
-------------------------------------------------
| 형식지정자| 36인경우 | -36인경우|
-------------------------------------------------
| %d | 36 | -36 |
| %i | 36 | -36 |
| %hd | 36 | -36 |
| %ld | 36 | -36 |
| %o | 44 | 37777777734 |
| %u | 36 | 4294967260 |
| %x | 24 | ffffffdc |
-------------------------------------------------
```

```
-------------------------------------------------
| 형식지정자 | 36인 경우 | -36인 경우 |
-------------------------------------------------
| %15d | 36 | -36 |
| %015d | 000000000000036 | -00000000000036 |
| % 15d | 36 | -36 |
| % 015d | 00000000000036 | -00000000000036 |
| %-15d | 36 | -36 |
| %- 15d | 36 | -36 |
| %15.4d | 0036 | -0036 |
| %-15.4d | 0036 | -0036 |
-------------------------------------------------
```

실수출력

형식 지정자	의미	출력 예
%f	소수점 고정 표기 형식으로 출력	123.456
%e	지수 표기 형식으로 출력, 지수 부분을 e로 표시	1.23456e+2
%E	지수 표기 형식으로 출력, 지수 부분을 E로 표시	1.23456E+2
%g	%e형식과 %f 형식 중 더 짧은 형식으로 출력	123.456
%G	%E형식과 %f 형식 중 더 짧은 형식으로 출력	123.456

```
-------------------------------------------------
| 형식지정자 | 23.567인 경우 | -23.567인 경우 |
-------------------------------------------------
| %15.2f | 23.57 | -23.57 |
| %015.2f | 000000000023.57 | -00000000023.57 |
| % 15.2f | 23.57 | -23.57 |
| %+15.4f | +23.5670 | -23.5670 |
| % 015.2f | 00000000023.57 | -00000000023.57 |
| %-15.2f | 23.57 | -23.57 |
| %- 15.2f | 23.57 | -23.57 |
| %-15.4f | 23.5670 | -23.5670 |
-------------------------------------------------
```

```
-------------------------------------------------
| 형식지정자 | 23.567인 경우 | -23.567인 경우 |
-------------------------------------------------
| %15.2e | 2.36e+001 | -2.36e+001 |
| %015.2e | 0000002.36e+001 | -000002.36e+001 |
| % 15.2e | 2.36e+001 | -2.36e+001 |
| %+15.4e | +2.3567e+001 | -2.3567e+001 |
| % 015.2e | 000002.36e+001 | -000002.36e+001 |
| %-15.2e | 2.36e+001 | -2.36e+001 |
| %- 15.2e | 2.36e+001 | -2.36e+001 |
| %-15.4e | 23.5670 | -2.3567e+001 |
-------------------------------------------------
```

문자와 문자열 출력

형식 지정자	의미	출력 예
%c	문자 출력	c
%s	문자열 출력	Hello World!

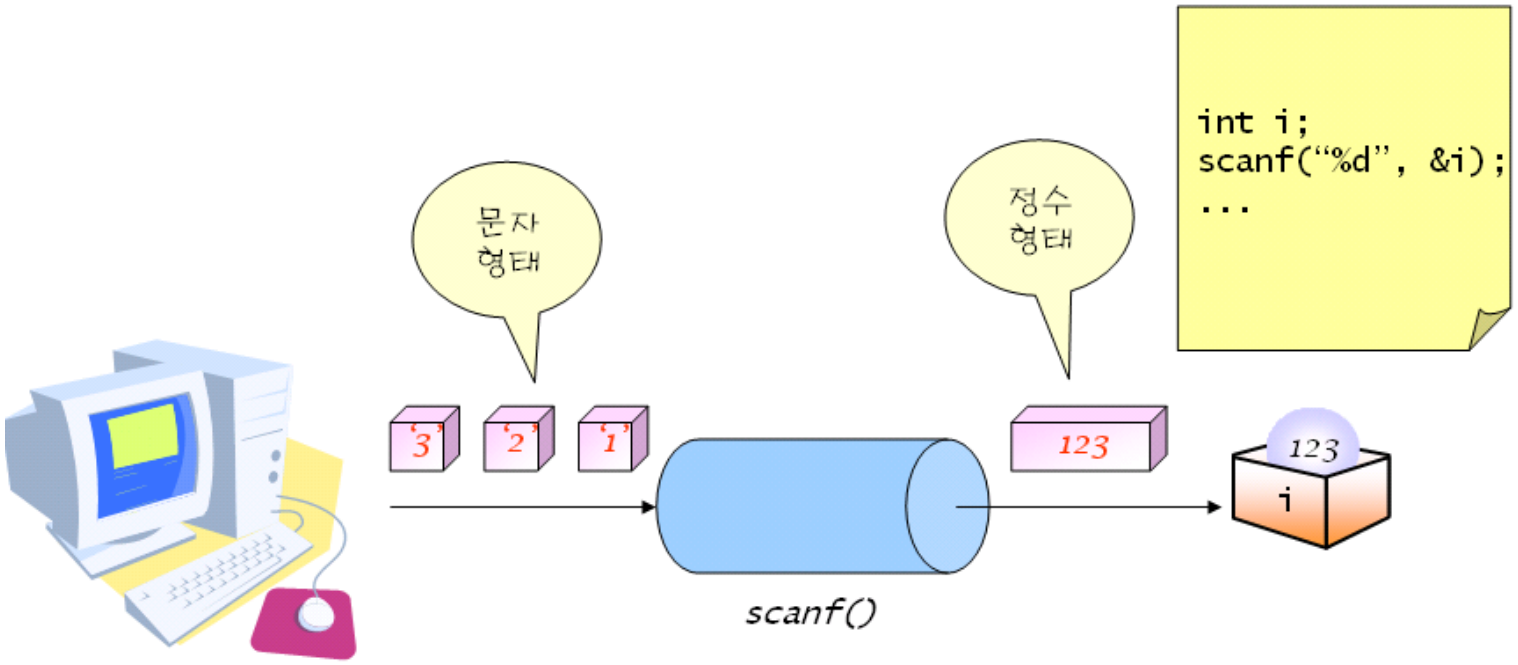
```
-------------------------------------------------
| 형식지정자 | 'b'인 경우 |
-------------------------------------------------
| %c | b
| %15c | b
| %015c | 000000000000000b
| %-15c | b
-------------------------------------------------
```

```
-------------------------------------------------
| 형식지정자 | "abcdefg"인 경우
-------------------------------------------------
| %s | abcdefg
| %15s | abcdefg
| %15.3s | abc
| %015s | 00000000abcdefg
| %-15s | abcdefg
-------------------------------------------------
```

제어 문자	의미
\a	벨소리(경고)
\b	백스페이스
\n	새로운 라인(new line)
\t	수평탭
\\	백슬래시
\?	의문부호
\'	홑 따옴표
\f	폼피드(form feed)
\"	이중 따옴표
\r	캐리지 리턴(carrage return)

scanf()를 이용한 입력

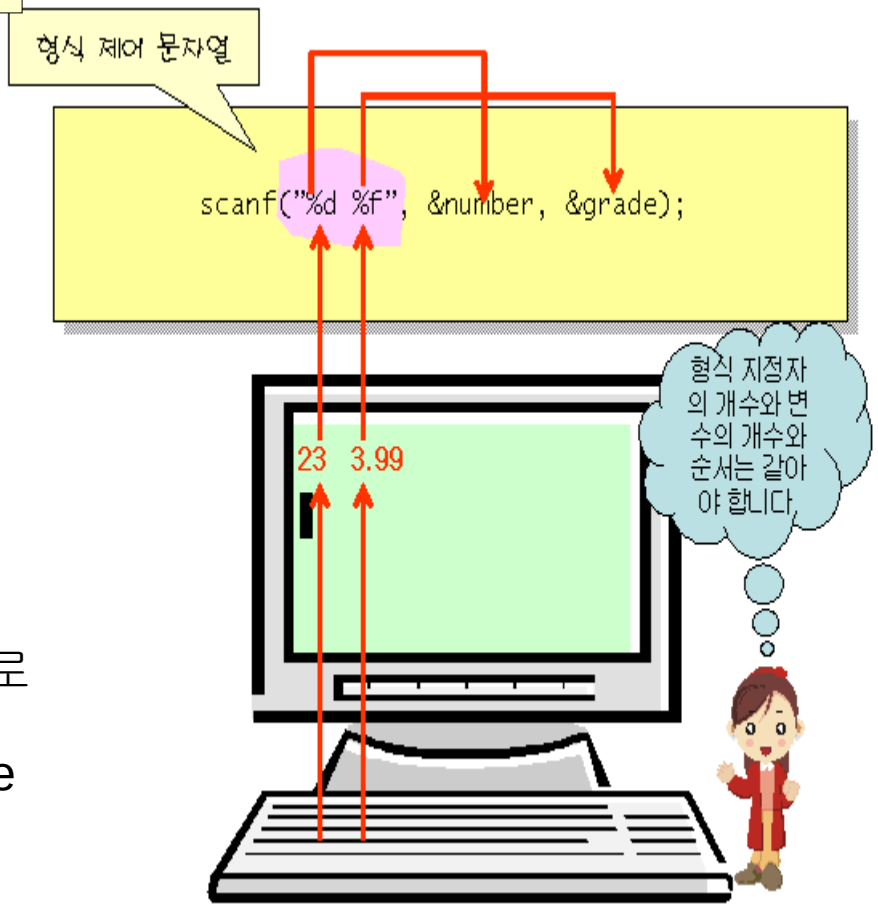
- 문자열 형태의 입력을 사용자가 원하는 형식으로 변환한다.



scanf()의 기본

%[*] [필드폭] [{h | l | L}] 형식

- *
  - 현재 입력을 무시하라는 의미
  - 파일에서 하나의 특정한 열만 읽을 때 유용
- 필드폭
  - 필드폭 만큼의 문자를 읽어서 값으로 변환
  - 공백 문자로 입력 값을 분리하지 않고서도 여러 개의 값들을 읽을 수 있다.
- 크기 지정
  - h가 정수형인 경우, short형으로 변환
  - h가 float형 앞에 붙으면 double형으로 변환
  - L은 long double형으로 변환



정수 입력

분류	형식 지정자	설명
정수형	%d	입력값을 int형으로 변환, 앞에 0이 붙으면 8진수로 가정, 앞에 0x가 붙으면 16진수로 가정한다.
	%u	부호없는 정수 형식으로 입력
	%o	입력을 8진수로 가정하고 정수로 변환
	%x	입력을 16진수로 가정하고 정수로 변환

```
#include <stdio.h>

int main(void)
{
        int d, o, x;

        scanf("%d %o %x", &d, &o, &x);
        printf("d=%d o=%d x=%d\n", d, o, x);

        return 0;
}
```

```
10
10
10
d=10 o=8 x=16
```

실수 입력

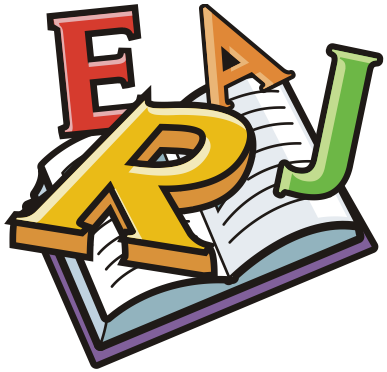
분류	형식 지정자	설명
정수형	%d	입력값을 int형으로 변환, 앞에 0이 붙으면 8진수로 가정, 앞에 0x가 붙으면 16진수로 가정한다.
	%u	부호없는 정수 형식으로 입력
	%o	입력을 8진수로 가정하고 정수로 변환
	%x	입력을 16진수로 가정하고 정수로 변환

```
#include <stdio.h>

int main(void)
{
        int d, o, x;

        scanf("%d %o %x", &d, &o, &x);
        printf("d=%d o=%d x=%d\n", d, o, x);

        return 0;
}
```

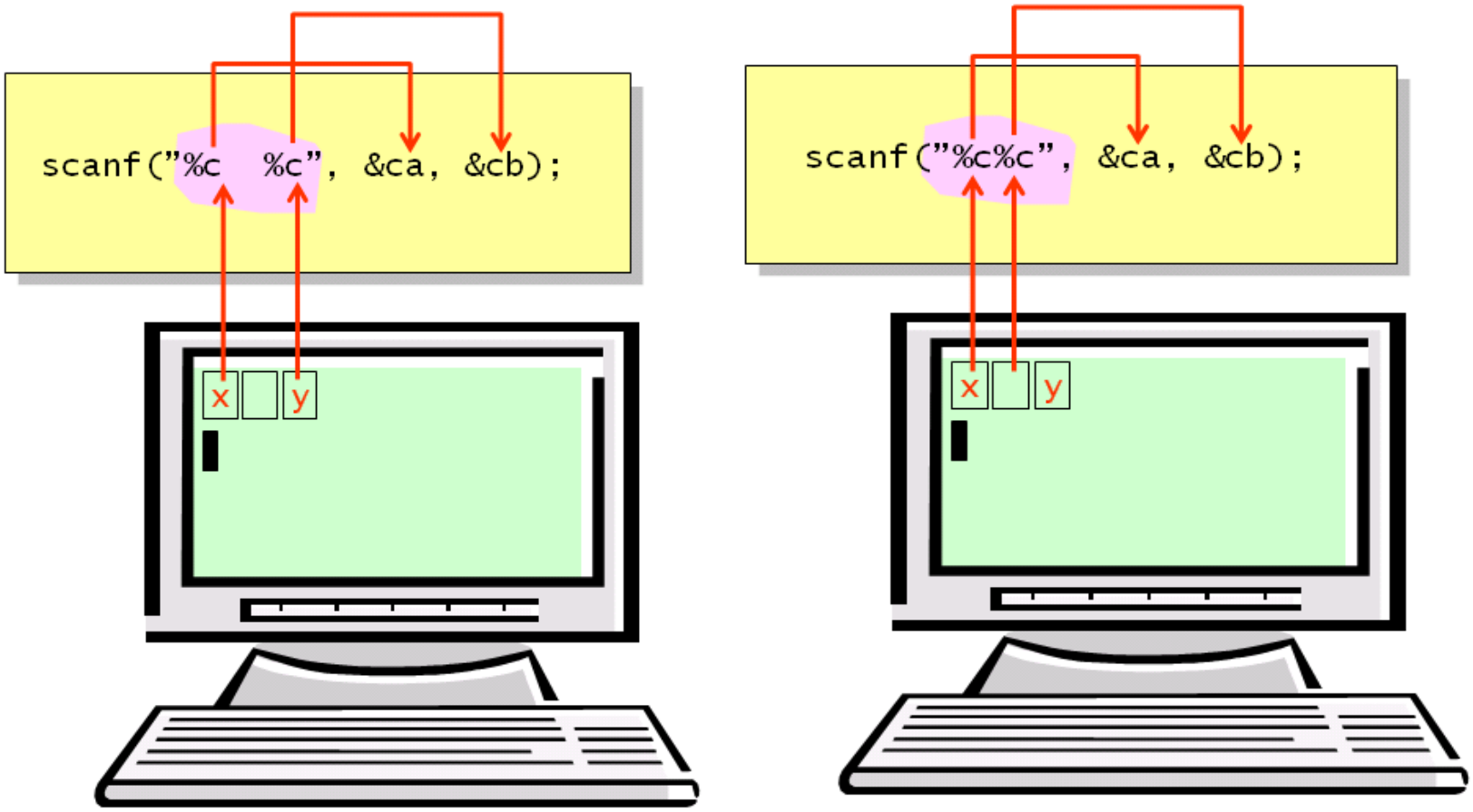
```
10
10
10
d=10 o=8 x=16
```

문자와 문자열 입력

분류	형식 지정자	설명
문자형	%c	char형으로 입력받음
	%s	공백 문자가 아닌 문자부터 공백 문자가 나올 때까지를 문자열로 변환하여 입력받음.
	%[abc]	대괄호 안에 있는 문자 a,b,c로만 이루어진 문자열을 읽어 들인다.
	%[^abc]	대괄호 안에 있는 문자 a,b,c만을 제외하고 다른 문자들로 이루어진 문자열을 읽어 들인다.
	%[0-9]	0에서 9까지의 범위에 있는 문자들로 이루어진 문자열을 읽어 들인다.



문자와 문자열 읽기



scanf6.c

```
1.  #include <stdio.h>

2.  int main(void)
3.  {
4.          char c;
5.          char s[80], t[80];

6.          printf("스페이스로 분리된 문자열을 입력하시오:");
7.          scanf("%s%c%s", s, &c, t);

8.          printf("입력된 첫번째 문자열=%s\n", s);
9.          printf("입력된 문자=%c\n", c);
10.         printf("입력된 두번째 문자열=%s\n", t);

11.         return 0;
12. }
```

```
스페이스로 분리된 문자열을 입력하시오:Hello World
입력된 첫번째 문자열=Hello
입력된 문자=
입력된 두번째 문자열=World
```

문자집합으로 읽기

```
1.  #include <stdio.h>

2.  int main(void)
3.  {
4.          char s[80];

5.          printf("문자열을 입력하시오:");
6.          scanf("%[abc]", s);

7.          printf("입력된 문자열=%s\n", s);

8.          return 0;
9.  }
```

```
문자열을 입력하시오:abcdef
입력된 문자열=abc
```

문자집합으로 읽기

```
1.  #include <stdio.h>

2.  int main(void)
3.  {
4.          char s[80];

5.          printf("문자열을 입력하시오:");
6.          scanf("%[a-z]", s);     // 알파벳 소문자(a-z)로 구성된 문자열만 입력
7.          printf("입력된 문자열=%s\n", s);

8.          return 0;
9.  }
```

```
문자열을 입력하시오:abcdefghijklmnOPQRSTUVWXYZ
입력된 문자열=abcdefghijklmn
```

특정 문자를 무시

```
1.  #include <stdio.h>

2.  int main(void)
3.  {
4.          int year, month, day;

5.          printf("날짜를 입력하시오: ");
6.          scanf("%d%*d%*d", &year, &month, &day);
7.          printf("입력된 날짜는 %d년 %d월 %d일입니다.\n", year, month, day);

8.          return 0;
9.  }
```

```
날짜를 입력하시오: 2007.9.1
입력된 날짜는 2007년 9월 1일입니다.
```

scanf() 사용시 주의점

- 입력값을 저장할 변수의 주소를 전달
  - int i;
  - scanf("%d", i); // 오류!!

- 배열의 이름은 배열을 가리키는 포인터
  - int str[80];
  - scanf("%s", str); // 올바름
  - scanf("%s", &str); // 오류!!

- 충분한 공간을 확보
  - int str[80];
  - scanf("%s", str); // 입력된 문자의 개수가 79를 초과하면 치명적인 오류 발생

- scanf()의 형식 제어 문자열의 끝에 줄바꿈 문자 '\n'을 사용하는 것은 해당 문자가 반드시 입력되어야 한다는 의미
  - scanf("%d\n", &i);// 잘못됨!!

memset()

- void *memset(void *dest, int c, size_t count);
  - dest가 가리키는 위치부터 n개만큼 c로 채운다

```
1.  #include <memory.h>
2.  #include <stdio.h>

3.  int main( void )
4.  {
5.          char buffer[] = "This is a test of the memset function";

6.          printf( "Before: %s\n", buffer );
7.          memset( buffer, '*', 4 );
8.          printf( "After:  %s\n", buffer );
9.
10.         return 0;
11. }
```

```
Before: This is a test of the memset function
After: **** is a test of the memset function
```

memcpy()

- void *memcpy(void *dest, const void *src, size_t count);
  - src 위치에서 count 개수 만큼 dest 위치로 복사한다
  - Src영역과 dest 영역이 겹칠 경우 어떻게 될지 정의하지 않는다

```
1.  #include <memory.h>
2.  #include <string.h>
3.  #include <stdio.h>

4.  char str1[7] = "aabbcc"

5.  int main( void )
6.  {
7.          printf( "The string: %s\n", str1 );
8.          memcpy( str1 + 2, str1, 4 );
9.          printf( "New string: %s\n", str1 );

10.         strcpy( str1, sizeof(str1), "aabbcc" );   // 문자열을 다시 초기화한다.

11.         printf( "The string: %s\n", str1 );
12.         memmove( str1 + 2, str1, 4 );
13.         printf( "New string: %s\n", str1 );

14.         return 0;
15. }
```

```
The string: aabbcc
New string: aaaabb ---- aaaaaa 이렇게 결과가 나올 수도 있다
The string: aabbcc
New string: aaaabb
```

memmove()

- void *memmove(void *dest, const void *src, size_t count);
  - src 위치에서 count 개수 만큼 dest 위치로 복사한다
  - src영역과 dest 영역이 겹칠 경우 겹치는 영역을 먼저 access하여 복사시키므로 정확한 복사가 이루어진다
  - 그러므로 겹치는 영역이 존재할 경우에는 정확한 복사를 위해서 memmove() 함수를 사용하는 것이 좋다

memcmp()

- int memcmp(const void *buf1, const void *buf2, size_t count);

```
1.  #include <string.h>
2.  #include <stdio.h>

3.  int main( void )
4.  {
5.          char first[]  = "12345678901234567890"
6.          char second[] = "12345678901234567891"
7.          int result;

8.          printf( "Compare '%.19s' to '%.19s':\n", first, second );
9.          result = memcmp( first, second, 19 );

10.         if( result < 0 )
11.                 printf( "First is less than second.\n" );
12.         else if( result == 0 )
13.                 printf( "First is equal to second.\n" );
14.         else
15.                 printf( "First is greater than second.\n" );
16. }
```

```
Compare '1234567890123456789' to '1234567890123456789':
First is equal to second.
```

memchr()

- void *memchr(const void *p, int c, size_t n)
  - p에서 시작하여 n개의 문자를 탐색하여 c와 일치하는 첫 번째 문자를 찾아서 그 주소를 리턴한다

라이브러리 함수 exit()

- exit()는 프로그램을 종료
- atexit()는 exit()가 호출되는 경우에 수행되는 함수들을 등록

```
1.  #include <stdlib.h>
2.  #include <stdio.h>

3.  void fn1( void ), fn2( void );

4.  int main( void )
5.  {
6.          atexit( fn1 );
7.          atexit( fn2 );
8.          printf( "프로그램이 종료되었습니다.\n" );
9.  }

10. void fn1()
11. {
12.         printf( "여기서 메모리 할당을 해제합니다.\n" );
13. }

14. void fn2()
15. {
16.         printf( "여기서 종료 안내 메시지를 내보냅니다.\n" );
17. }
```

```
프로그램이 종료되었습니다.
여기서 종료 안내 메시지를 내보냅니다.
여기서 메모리 할당을 해제합니다.
```

라이브러리 함수 qsort()

- 퀵정렬을 수행하는 라이브러리 함수
- **void** qsort(**void** *base, size_t num, size_t width, **int** (*compare)(const **void** *, const **void** *))

- compare((**void** *) elem1, (**void** *) elem2);

반환값	설명
< 0	elem1이 elem2보다 작으면
0	elem1이 elem2과 같으면
> 0	elem1이 elem2보다 크면

quick sort

- Divide and Conquer 방법을 사용
  - 주어진 리스트 가운데 하나의 원소를 선택한다 : pivot
  - pivot 앞에는 pivot보다 작은 원소들이 위치하고, 뒤에는 작은 원소들이 위치하도록 리스트를 분할한다. 분할이 끝나면 pivot의 위치가 확정된다
  - 분할된 두 개의 리스트에 대해 리스트의 크기가 0이나 1이 될때까지 동일한 과정을 반복한다

피벗은 p, 리스트 왼쪽과 오른쪽 끝에서 시작한 인덱스들을 i,j라고 하자.

5 - 3 - 7 - 6 - 2 - 1 - 4

p

리스트 왼쪽에 있는 i 위치의 값이 피벗 값보다 크고, 오른쪽에 있는 j 위치의 값은 피벗 값보다 작으므로 둘을 교환한다.

5 - 3 - 7 - 6 - 2 - 1 – 4

i j p

1 - 3 - 7 - 6 - 2 - 5 – 4

i j p

j 위치의 값이 피벗 값보다 작지만, i 위치의 값도 피벗값보다 작으므로 교환하지 않는다.

1 - 3 - 7 - 6 - 2 - 5 – 4

i j p

i위치를 피벗 값보다 큰 값이 나올 때 까지 진행해 j 위치의 값과 교환한다.

1 - 3 - 7 - 6 - 2 - 5 – 4

i j p

1 - 3 - 2 - 6 - 7 - 5 - 4

i j p

i위치와 j 위치의 값이 만나면, j 위치의 값과 피벗 값을 교환한다.

1 - 3 - 2 - 6 - 7 - 5 - 4

p

1 - 3 - 2 - 4 - 7 - 5 - 6

p

피벗 값 좌우의 리스트에 대해 각각 퀵 정렬을 재귀적으로 수행한다.

1 - 3 - 2

1 - 2 - 3

완성된 리스트는 다음과 같다.

1 - 2 - 3 - 4 - 5 - 6 - 7

qsort()

```
1.  #include <stdlib.h>
2.  #include <string.h>
3.  #include <stdio.h>

4.  //
5.  int compare( const void *arg1, const void *arg2 )
6.  {
7.          if( *(double *)arg1 > *(double *)arg2 ) return 1;
8.          else if( *(double *)arg1 == *(double *)arg2 ) return 0;
9.          else return -1;
10. }
11. //
12. int main(void)
13. {
14.         int i;
15.         double list[5] = {2.1, 0.9, 1.6, 3.8, 1.2};

16.         qsort( (void *)list, (size_t)5, sizeof(double), compare );

17.         for(i=0;i<5;i++)
18.                 printf("%f ", list[i]);

19.         return 0;
20. }
```

```
0.900000 1.200000 1.600000 2.100000 3.800000
```

```
1.  #include <stdlib.h>
2.  #include <string.h>
3.  #include <stdio.h>
    # define N   11
    enum when {before, after};
1.  int compare( const void *arg1, const void *arg2 ) {
2.          if( *(double *)arg1 > *(double *)arg2 ) return 1;
3.          else if( *(double *)arg1 == *(double *)arg2 ) return 0;
4.          else return -1;
5.  }
6.  int main(void) {

7.          double list[N];
8.          fillArray(list, N);
9.          printArray(before, list, N);
10.         qsort( (void *)list, N, sizeof(double), compare );
11.          printArray(after, a, N);
12.         return 0; }
13. void printArray(when val, double *a, int n) {
14.   int  i;
    printf("%s\n%s%s\n", "---", ((val == before) ? "Before " : "After "), "sorting:");
    for(i=0; i<n; i++) {
      if(i%6 == 0) putchar('\n');
      printf("%10.1f", a[i]); }
    putchar('\n'); }

15. void fillArray(double *a, int n) {
    int   I;
    srand(tinme(NULL));
    for(i=0; i<n; i++)
          a[i] = (rand()%1001) / 10.0; }
```

Searching binary tree

- breadth first search
- depth first search

bsearch() : 이진탐색

- void *bsearch(const void *key, const void *base, size_t number, size_t size, int (*compare)(const void *, const void *))
- 전제조건 : 탐색대상이 정렬되어 있어야 한다
  - 이 경우 1,000개의 엘리먼트에 대해 단지 10번의 비교만 이루어진다

질문???