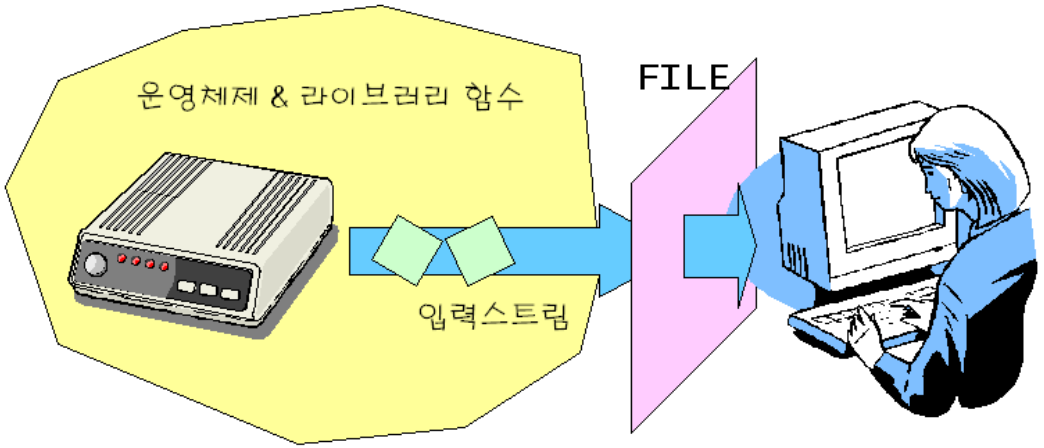
입출력 라이브러리 함수

스트림과 파일

- 스트림은 구체적으로 FILE 구조체를 통하여 형성
- 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;
```



입출력 함수의 분류

스트림 형식	표준 스트림	파일 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(파일 스트림)
- 데이터의 형식에 따른 분류
  - getchar()나 putchar()처럼 *문자* 형태의 데이터를 받아들이는 입출력
  - printf()나 scanf()처럼 구체적인 형식을 지정할 수 있는 입출력

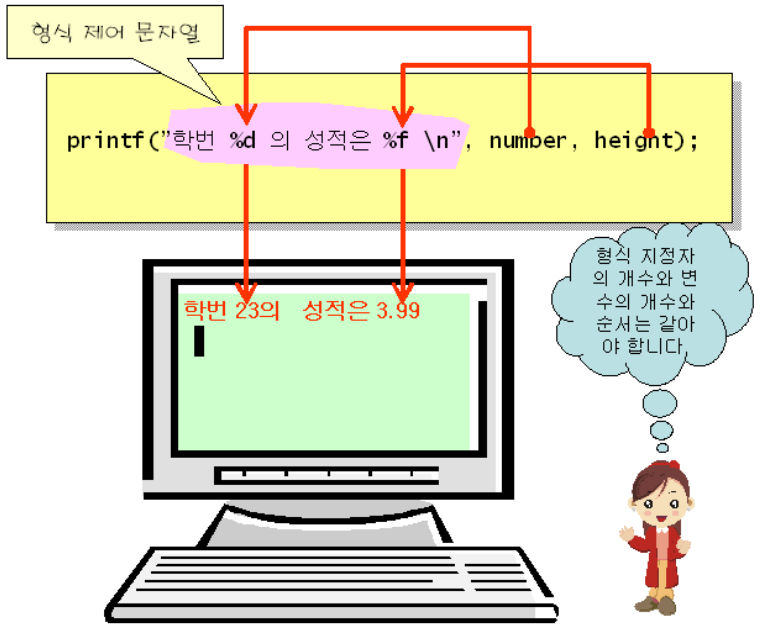
printf()를 이용한 출력

```
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.



플래그(flag)

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

출력문	출력결과	설명
printf("%d", 123)	123	정수의 기본 출력
printf("%010d", 123)	0000000123	10자리중 빈공간은 0으로 채움(오른쪽 정렬)
printf("%-10d", 123)	123bbbbbbb	왼쪽 정렬(0으로 채우지 않으나 공백7자리)
printf("% d", 123)	123	양수이면 빈칸 음수이면 – 를 붙인다
printf("%+d", 123)	+123	양수이면 + 음수이면 – 를 붙인다
printf("%10d", 123)	bbbbbbb123	10자리로 출력하되 오른쪽 정렬
printf("%#x", 16)	0x10	앞에 0x를 붙여 16진수로 출력
printf("%#o", 15)	017	앞에 0을 붙여 8진수 출력

필드폭과 정밀도

- 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

- 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

변환문자: 정수 및 실수 출력

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

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

int i = 123 ;
double x = 0.123456789 ;

형식	출력	설명
printf("%d", i)	"123"	정수 기본 출력
printf("%05d", i)	" 00123"	필드폭 = 5 (0으로 채움, 오른쪽 정렬)
printf("%7o", i)	" 123"	필드폭 = 7 (8진수 0으로 채우지 않으나 공백4자리)
printf("%-9x", i)	"7b "	왼쪽 정렬 16진수 공백7자리
printf("%-#9x", i)	"0x7b "	왼쪽 정렬 16진수(0x표기 16진수)
printf("%10.5f", x)	" 0.123456"	필드폭 = 10자리, 정밀도 5자리
printf("%-12.5e", x)	"1.23457e-01 "	필드폭 = 12자리, 정밀도 5자리

문자와 문자열 출력

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

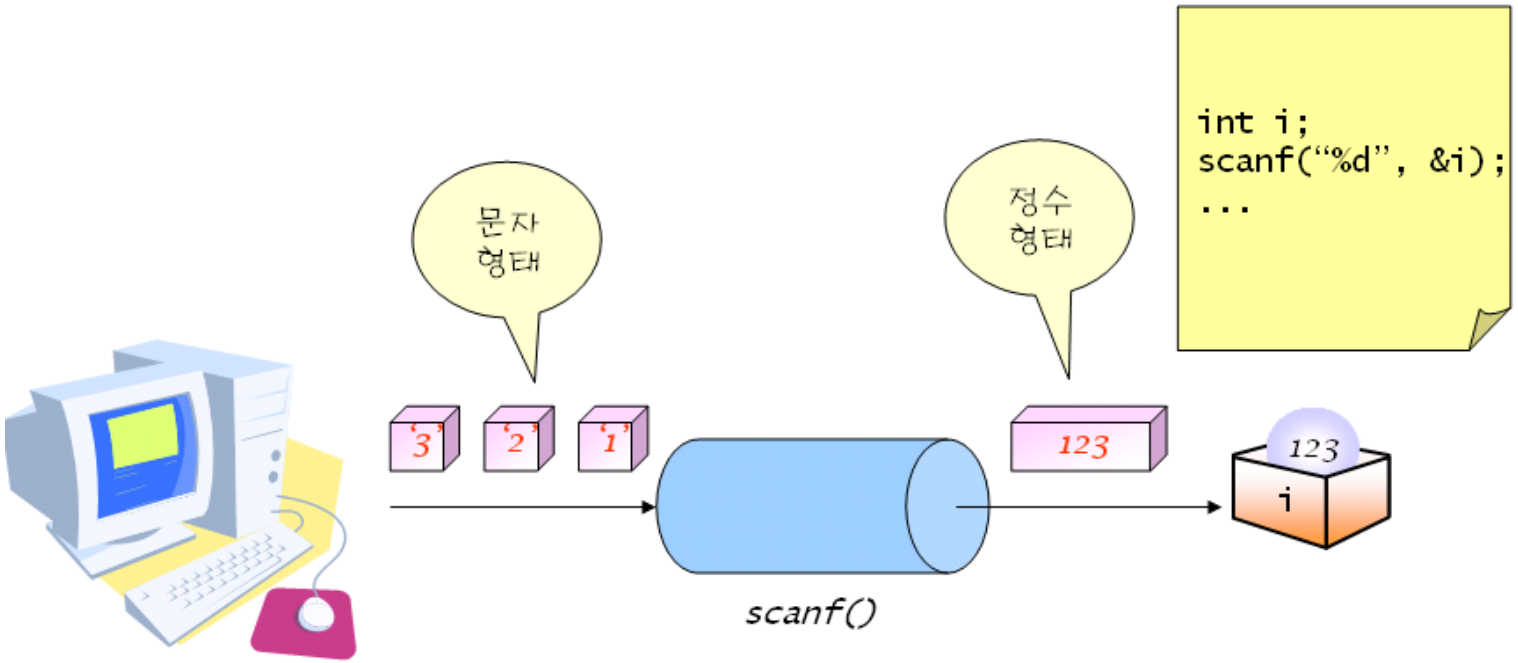
char c = 'A', s[] "Blue moon!" ;

형식	출력	설명
printf("%c", c)	"A"	문자 기본 출력
printf("%2c", c)	" A"	필드폭 = 2, 오른쪽 정렬
printf("%-3c", c)	"A "	필드폭 = 4. 왼쪽 정렬
printf("%s", s)	"Blue moon!"	문자열 기본 출력
printf("%3s", s)	"Blue moon!"	문자열 부족하나 모두 출력
printf("%.6s", s)	"Blue m"	정밀도 6자리
printf("%-11.8s", s)	"Blue moo "	정밀도 8자리, 왼쪽 정렬

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

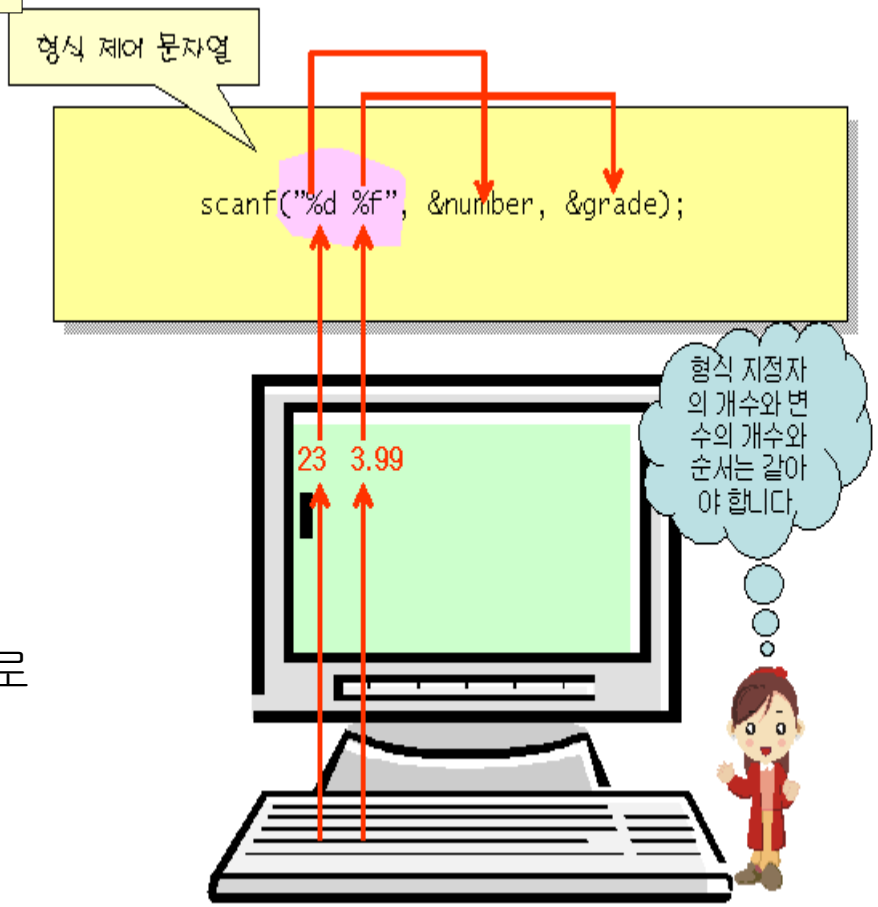
- 문자열 형태의 입력을 사용자가 원하는 형식으로 변환하여 저장



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
  - l이 가 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
```

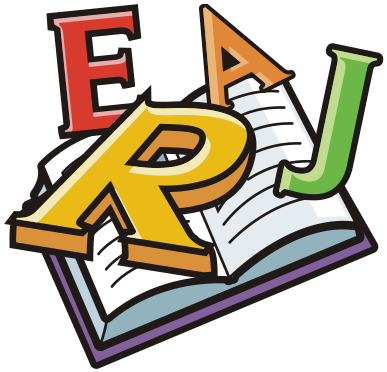
실수 입력

분류	형식 지정자	설명
실수형	%f	입력값을 실수형으로 변환
	%e	입력값을 부둥소숫점 e-format으로 변환
	%g	입력값을 f와 e 형식 중 짧은 것으로 변환
	%E, %G	%e와 %g와 동일하나 대문자를 사용

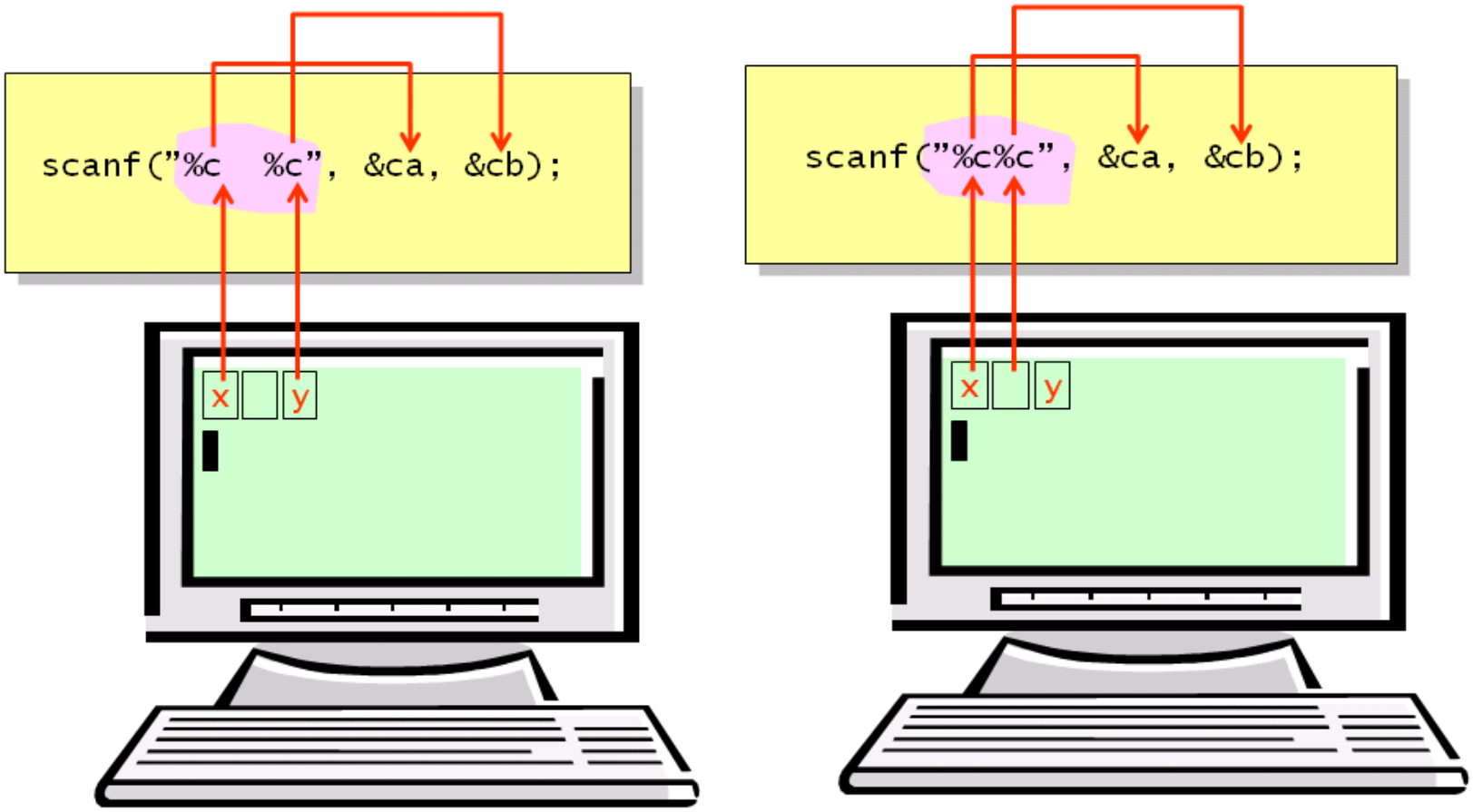
<printf(…) 문을 참조>

문자와 문자열 입력

분류	형식 지정자	설명
문자형	%c	한 개의 문자 입력
	%s	공백 문자가 아닌 문자부터 공백 문자가 나올 때까지를 문자열로 변환하여 입력
	%[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. 그러므로 a, b, c 를 만나면 입력 종료
	%[0-9]	0에서 9까지의 범위에 있는 문자들로 이루어진 문자열을 읽어 들인다.



문자와 문자열 읽기



예제

```
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.
```

memset()

- void *memset(void *dest, int c, size_t n);
  - 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.
```