컴퓨터 프로그래밍

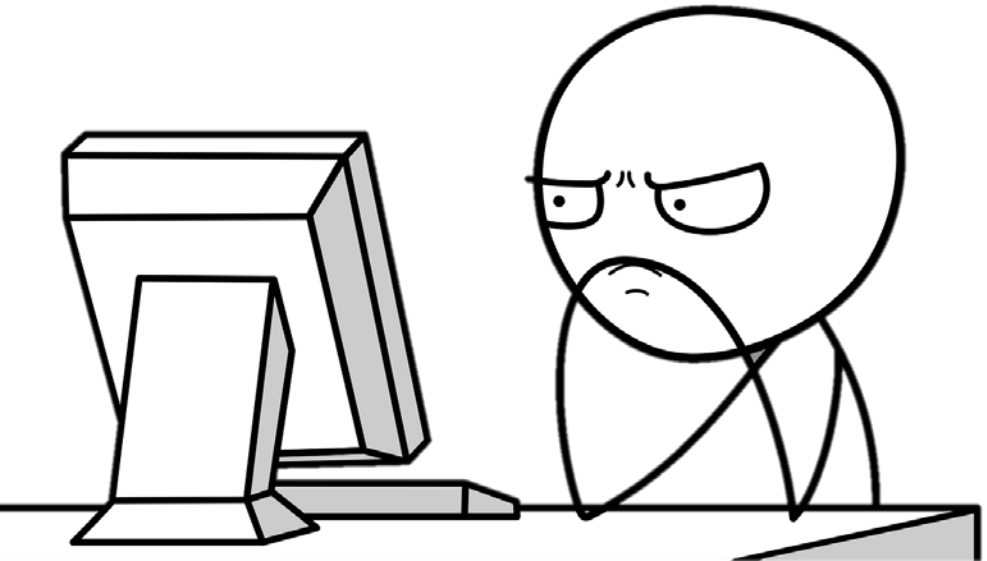
Computer Programming

04
전처리와 입출력

목차

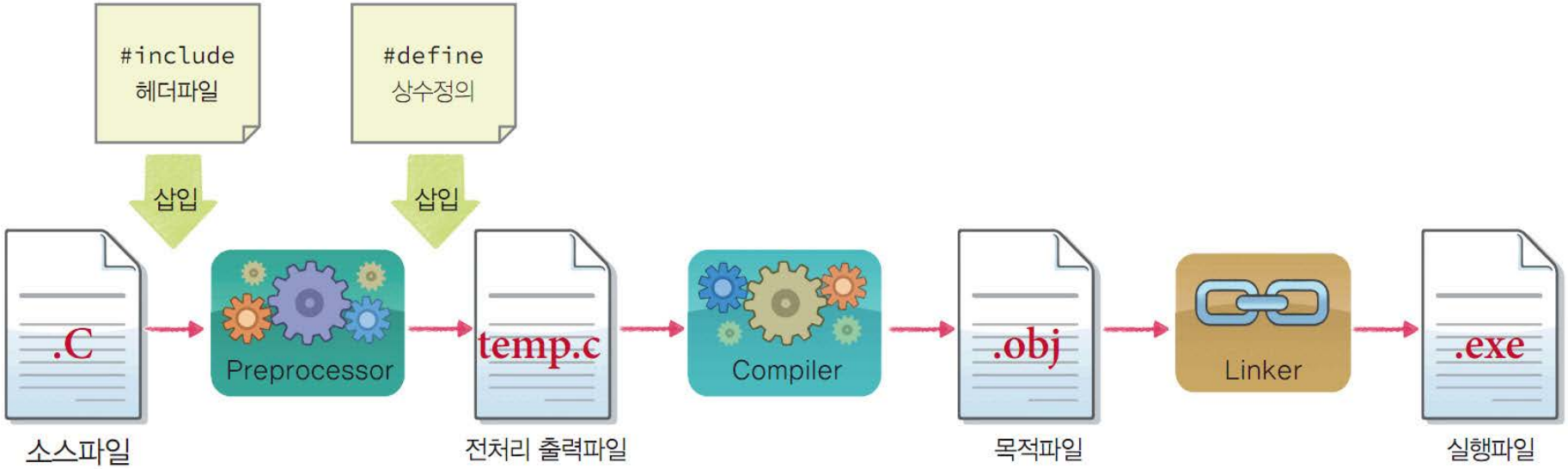
1. 전처리
2. 출력 함수 printf()
3. 입력 함수 scanf()

1. 전처리



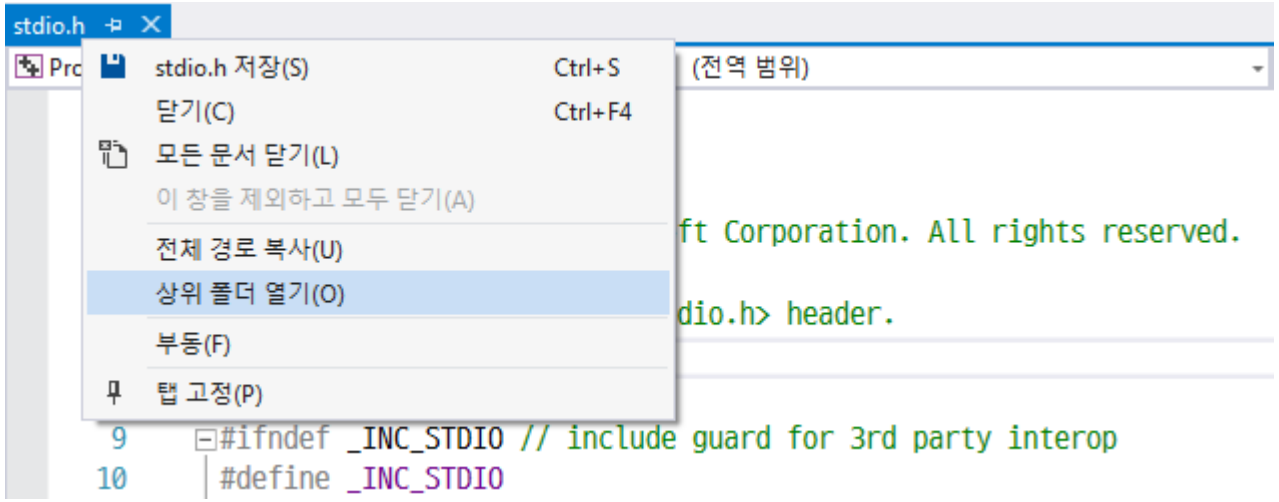
전처리 개요

- 전처리기 역할
  - 컴파일compile 전에 전처리기preprocessor의 전처리preprocess 과정이 필요
  - 결과인 전처리 출력파일을 만들어 컴파일러에게 보내는 작업을 수행
- 전처리 지시자preprocess directives
  - #include, #define과 같은 전처리 지시자는 항상 #으로 시작
    - 마지막에 세미콜론 ; 이 없는 등 일반 C 언어 문장과는 구별
  - 조건 지시자로 #if, #elif, #else, #endif, #ifdef, #ifndef, #undef 등



전처리 지시자 #include

- 헤더파일
  - #include <stdio.h>
    - #include, #define 등
    - 자료형의 재정의typedef, 함수원형prototype 정의 등과 같은 문장이 있는 텍스트 파일
  - 대표적인 헤더파일, 확장자 *.h
    - stdio.h
  - 헤더파일 직접 보기



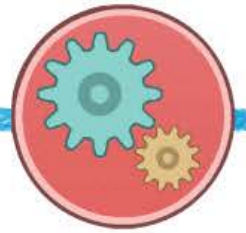
전처리 지시자 #include

소스파일

```
#include <stdio.h>

int main(void)
{
    ...
    printf("원면적: %f\n", area);

    return 0;
}
```

삽입

헤더파일
stdio.h

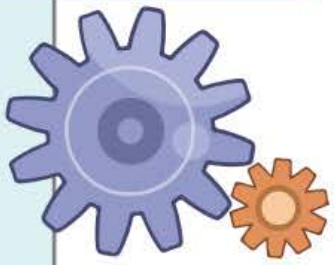


전처리기

전처리 과정을 거친 임시 소스파일

```
/***
*stdio.h - definitions/declarations
for standard I/O routines
...

int main(void)
{
    ...
    printf("원면적: %f\n", area);

    return 0;
}
```

파일 stdio.h에 함수 printf()를 위한 함수원형이 선언되어 있으므로 printf() 사용이 가능함.



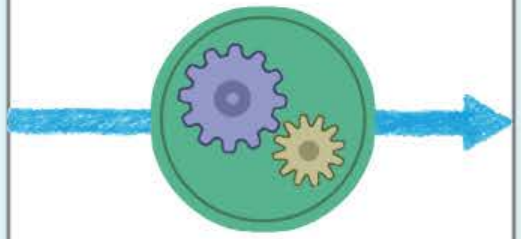
전처리 지시자 #define

- 매크로 상수
  - 전처리기preprocessor는 소스에서 정의된 매크로 상수를 모두 #define 지시자에서 정의된 문자열로 대체(replace)
- #define identifier_name [value]
  - #define에 정의된 identifier_name은 전처리기에 의해 모두 value로 대체되어 컴파일
  - #define은 정수, 실수 또는 문자열 등의 상수를 KPOP, PI, PRT 등의 이름으로 정의
  - PI라는 매크로 상수
    - 전처리 과정에서 모두 3.14라는 실수로 값이 바뀐 소스로 컴파일
  - 단 매크로 상수는 문자열 내부 또는 주석 부분에서는 대체되지 않음

전처리 지시자 #define

```
#define   KPOP 50000000          //정수 매크로 상수
#define   PI 3.14                //실수 매크로 상수
#define   PRT printf("종료\n")   //문자열 매크로 상수
```

```
#include <stdio.h>

#define PI 3.14

int main(void)
{
   double radius = 2.83;
   double area = radius * radius * PI;
   printf("원면적: %f\n", area);

   return 0;
}
```




```
#include <stdio.h>


int main(void)
{
   double radius = 2.83;
   double area = radius * radius * 3.14;
   printf("원면적: %f\n", area);

   return 0;
}
```

Source Code #01: advancemacro.c

```c
// file: advancemacro.c

#include <stdio.h>

#define MESSAGE "프로그램언어의 학습은 일반언어의 학습과 \
                 같이 반복학습이 중요하다"

#define PI          3.141592                //PI를 3.14로 대치하는 지시자
#define VOLUME(r)   (4 * PI * CUBE(r) / 3)  //구의 체적를 구하는 매크로
#define SQUARE(x)   ( (x) * (x) )           //인자 x의 제곱 구하는 매크로
#define CUBE(x)     ( SQUARE(x) * (x) )     //인자 x의 세제곱 구하는 매크로
#define MULT(x, y)  ( (x) * (y) )           //인자 x, y의 곱 구하는 매크로

int main(void)
{
    double radius = 2.32;
    printf("반지름이 %.2lf인 구의 체적은 %.2lf 입니다.\n", radius, VOLUME(radius));
    printf("실수 %.2f의 제곱은 %.2f 입니다.\n", 4.29, SQUARE(4.29));
    printf("실수 %.2f의 제곱은 %.2f 입니다.\n", 3.0, CUBE(3.0));
    printf("실수 %.2f와 실수 %.2f의 곱은 %.2f입니다.\n", 2.78, 3.62, MULT(2.78, 3.62));
    puts(MESSAGE);

    return 0;
}
```

- #define에서 그 활용도를 높이기 위한 방안이 함수와 같이 인자parameter를 이용하는 방법

```
반지름이 2.32인 구의 체적은 52.31 입니다.
실수 4.29의 제곱은 18.40 입니다.
실수 3.00의 제곱은 27.00 입니다.
실수 2.78와 실수 3.62의 곱은 10.06입니다.
프로그램언어의 학습은 일반언어의 학습과
같이 반복학습이 중요하다
```

인자를 사용한 매크로

인자인 x부분이 실제 사용한 수로 대체된다.

```
#define SQUARE(x)        ( (x) * (x) )

printf("%d, %d\n", SQUARE(2), SQUARE(3));
```


```
printf("%d, %d\n", ( (2)*(2) ), ( (3)*(3) ));
```

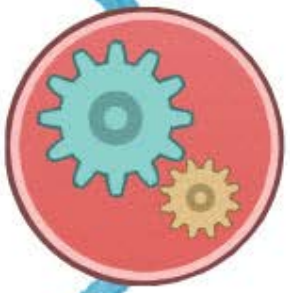
인자를 사용한 매크로 주의점

- 매크로를 구성하는 모든 인자와 외부에 괄호를 이용

```
#define SQUARE(x)      ( x * x )

...
int a = SQUARE(1+2);
```

인자인 x부분이1+2로 대체되어
a에는 실제 5가 저장된다.

전처리기



```
int a = ( 1+2 * 1+2 );
```

인자를 사용한 매크로 주의점

- 기호 상수에서 매크로 이름과 시작괄호 사이에는 공백이 올 수 없음

매크로 이름과 시작괄호 사이에는
공백이 허용되지 않는다.

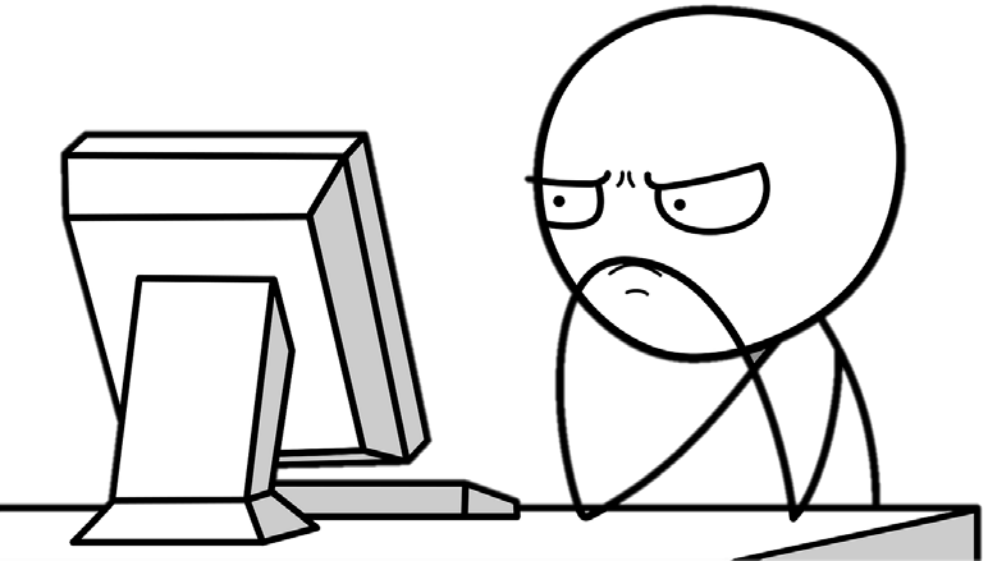
```
#define SQUARE (x) ( (x) * (x) )        //잘못된 매크로 정의
#define SQUARE(x)     ( (x) * (x) )
#define CUBE(x)       ( SQUARE(x) * (x) )
```

Lab #01: 문자열 출력을 위한 매크로 정의

- 다음 정보를 이용하여 매크로 myprint(x)를 정의해 인자인 문자열 x를 한 행에 출력하는 프로그램 작성
  - 전처리 지시자 #define으로 인자가 있는 매크로 myprint(x)를 정의
- 출력
  - 매크로로 출력하기
  - 출력함수로 출력하기

```c
// file: basicmacro.h

#include <stdio.h>

#define myprint(x)                \
                                  


int main(void)
{
    myprint("매크로로 출력하기");
    printf("출력함수로 출력하기\n");

    return 0;
}
```

2. 출력함수 printf()



printf()의 인자

- 형식문자열과 출력할 목록으로 구분
  - 형식문자열에서 %d와 같이 %로 시작하는 형식지정자 순서대로 서식화하여 그 위치에 출력

```
int TGERM = 15
    printf(" %d 의 두 배는 %d 입니다.\n", term, 2*term);
```

형식문자열 출력목록

함수 printf()의 첫 번째 인자인 형식문자열(format string)

- 일반문자와 이스케이프 문자
  - 이스케이프 문자는 \"와 \'같이 \로 시작하는 문자
- 형식 지정자format specification로 구성
  - %d와 %s와 같이 %로 시작하는 형식지정자
- 형식지정자 %d 위치에 바로 20이라는 정수가 출력
- 이스케이프 문자 \"는 문자 "이 그대로 출력

```
printf("%d대 연애에서 가장 중요한 것은 \"밀당\"이다.", 20);
```

함수 printf()에서 정수 출력을 위한 형식 지정자

- 정수의 십진수 출력을 위한 형식 지정자는 %d와 %i
- 8진수로 출력하려면 %o를 이용
  - 앞 부분에 숫자 0이 붙는 출력을 하려면 %#o를 이용
- 소문자의 십육진수로 출력하려면 %x와 대문자로 출력하려면 %X를 이용
  - 16진수 앞에 0x또는 0X를 붙여 출력하려면 #을 삽입하여 %#x와 %#X를 이용
- 함수 printf()
  - 반환값은 출력한 문자 수이며, 오류가 발생하면 음수를 반환

함수 printf()에서 정수 출력을 위한 형식 지정자

형식문자열(format)에서 형식지정자(%d 등)에 맞게 두 번째 인자인 arg1부터 형식 지정자 위치에 출력하며, 형식 지정자를 제외한 일반 문자와 이스케이프 문자는 그 위치에 그대로 출력된다.

char *format | arg1 | arg2 | ...

```
printf("...%문자...%문자... %문자...", 변수명 또는연산식, 변수명 또는연산식, ...)
```

함수 printf()로 콘솔에 출력

```
int n = 255;
printf("저장값: %d %i %o %#o %#x %#X\n", n, n, n, n, n, n);
```

출력:

```
저장값: 255 255 377 0377 0xff 0XFF
```

실수를 위한 출력 %f

- 실수의 간단한 출력을 위한 형식 지정자는 %f
  - 형식지정자 %f는 실수를 기본적으로 3.400000와 같이 소수점 6자리까지 출력
  - 함수 printf()에서 실수 출력으로 %f와 함께 %lf도 사용
- 출력 폭의 지정
  - 출력 필드 폭이 출력 내용의 폭보다 넓으면 정렬은 기본이 오른쪽, 필요하면 왼쪽으로 지정

폭과 정렬 형식 지정자

+는 부호를 출력시키며, 전체 폭이 출력될 값 폭보다 크면 우측정렬이 기본이고, -를 기술하면 좌측정렬로 출력됨.

전체 폭은 소수점과 소수점 이하 폭, 그리고 정수 부분이 출력될 폭의 합보다 커야 함. 전체 폭이 출력될 값의 폭보다 작으면 전체 폭을 무시하고 원래대로 출력됨.

%[+|-][전체폭].[소수점이하폭]{d|i|f}

`printf("%d\n", n);`	255	정수를 있는 위치에 그대로 출력
`printf("%10d\n", n);`	255	폭이 10이며 우측정렬로 출력
`printf("%+10d\n", n);`	+255	+에 의해 부호 +가 출력
`printf("%-10d\n", n);`	255	-에 의해 좌측정렬로 출력
`printf("%f\n", f);`	3.141592	실수는 소수점 이하 6자리까지 출력
`printf("%10f\n", f);`	3.141592	폭이 10이며 우측정렬로 출력
`printf("%10.4f\n", f);`	3.1416	폭이 10이며 소수점 이하 4자리까지 우측정렬로 출력
`printf("%+10.4f\n", f);`	+3.1416	+에 의해 부호 +가 출력
`printf("%-10.4f\n", f);`	3.1416	-에 의해 좌측정렬로 출력

형식 지정자 정리

- %로 시작하는 형식 지정자는 %와 서식문자 d 사이
- %[flags][width].[precision]{h|l}d와 같이 여러 종류의 지정자가 가능
- 정수를 위한 형식 지정자 d바로 앞에 short을 의미하는 h와 long을 의미하는 l, 그리고 long long을 의미하는 ll이 가능
- 실수를 위한 형식 지정자 f바로 앞은 double을 의미하는 l이 가능하나, %f와 %lf는 차이가 없음
- 옵션 [flags]로는 정렬, 부호표시 등을 지정
- 옵션 [width].[precision]으로 출력부분의 전체 폭과 소수점 이하 자릿수 폭을 지정

형식 지정자 정리

-, +, 0, # 등이 올 수 있다.

10.3과 같이 정수로 출력 폭을 지정한다.

```
%[flags][width].[precision]{h|l|ll}type
```

예: %+10.3lf, %-10hd, %#10X

인자의 크기를 지정하는 옵션으로 서식문자 d 앞의 h는 short를, l은 long을, ll은 long long을 의미한다.

형식문자type field characters 종류

서식문자	자료형	출력 양식
c	char, int	문자 출력
d, i	int	부호 있는 정수 출력으로, lf는 long int, lld는 long long int형 출력
o	unsigned int	부호 없는 팔진수로 출력
x, X	unsigned int	부호 없는 십육진수 출력 x는 3ff와 같이 소문자 십육진수로, X는 3FF와 같이 대문자로 출력, 기본으로 앞에 0이나 0x, 0X는 표시되지 않으나 #이 앞에 나오면 출력
u	unsigned int	부호 없는 십진수(unsigned decimal integer)로 출력
e, E	double	기본으로 m.ddddddExxx의 지수 형식 출력(정수 1자리와 소수점 이하 6자리, 지수승 3자리), 즉 123456.789이라면 1.234568e+005로 출력
f, lf	double	소수 형식 출력으로 m.123456 처럼 기본으로 소수점 6자리 출력되며, 정밀도에 의해 지정 가능, lf는 long double 출력
g, G	double	주어진 지수 형식의 실수를 e(E) 형식과 f 형식 중에서 짧은 형태(지수가 주어진 정밀도 이상이거나 -4보다 작으면 e나 E 사용하고, 아니면 f를 사용)로 출력, G를 사용하면 E가 대문자로
s	char *	문자열에서 '\o'가 나올 때 까지 출력되거나 정밀도에 의해 주어진 문자 수만큼 출력
p	void *	주소값을 십육진수 형태로 출력
%		%를 출력

옵션 지정 문자flags 종류

문자	기본(없으면)	의미	예와 설명
-	우측정렬	수는 지정된 폭에서 좌측정렬	%-10d
+	음수일 때만 - 표시	결과가 부호가 있는 수이면 부호 +, -를 표시	%+10d
0	0을 안 채움	우측정렬인 경우, 폭이 남으면 수 앞을 모두 0으로 채움	%010x %-0처럼 좌측정렬과 0 채움은 함께 기술해도 의미가 없음
#	리딩 문자 0, 0x, 0X가 없음	서식문자가 o(서식문자 octal)인 경우 0이 앞에 붙고, x(서식문자 heXa)인 경우 ox가 붙으며, X인 경우 0X가 앞에 붙음	수에 앞에 붙는 0이나 0x는 0으로 채워지는 앞 부분에 출력

Source Code #02: specification.c

```c
// file: specification.c

#include <stdio.h>

int main(void)
{
    printf("%010d %s\n", 12345, "%010d");
    printf("%+010d %s\n\n", 1234, "%+010d");
    printf("%10o %s\n", 271, "%10o");
    printf("%0#10o %s\n", 271, "0#10o");
    printf("%-#10o %s\n\n", 271, "%-#10o");
    printf("%0#10x %s\n", 271, "0#10x");
    printf("%-#10X %s\n\n", 271, "%-#10X");

    printf("%d %s\n", 32768, "%d");
    printf("%hd %s\n", 32768, "%hd");

    return 0;
}
```

- %[flags][width].[precision]{h|l}d
- %[flags][width].[precision]o
- %[flags][width].[precision]x
- %[flags][width].[precision]f

```
0000012345 %010d
+000001234 %+010d

       417 %10o
0000000417 0#10o
0417       %-#10o

0x0000010f 0#10x
0X10F      %-#10X

32768 %d
-32768 %hd
```

정수의 다양한 출력

형식지정자	정수	결과
%d	1234	1234
%6i	1234	__1234
%+6i	1234	+_1234
%-06i	1234	+01234
%60	037	____37
%-60	037	37

형식지정자	정수	결과
%-#6o	037	037___
%#-6o	037	037___
%05x	0x1f	0001f
%0#6x	0x1f	0x001f
%-0#5x	0x1f	0x1f_
%#6X	0x1f	__0X1F

Source Code #03: stringprint.c

```c
// file: stringprint.c

#include <stdio.h>

int main(void)
{
    printf("사계절은 봄 여름 가을 겨울이다.\n");
    printf("사계절은 %s %s %s %s이다.\n\n", "봄", "여름", "가을", "겨울");
    printf("%s\n", "123456789012345");
    printf("%10.3s\n", "Hello!");
    printf("%-10.3s\n", "Hello!");
    printf("%4s\n", "Hello!");
    printf("%10.*s\n\n", 5, "Hello!");

    printf("%s\n", "123456789012345");
    printf("%s\n", "Hi, C language!");
    //전체폭 10에서 3개의 문자만 출력, 기본이 오른쪽 정렬
    printf("%10.3s\n", "Hi, C language!");
    //전체폭 10에서 3개의 문자만 출력, -는 왼쪽 정렬
    printf("%-10.3s\n", "Hi, C language!");
    //*는 정밀도를 입력으로 받아 지정, 정밀도가 3이므로 %10.3f로 출력
    printf("%10.*f\n", 3, 124.56789);

    //형식 문자열 내부에서는 %%가 % 출력
    printf("%10.2f%%\n", 3.25);
    //문자열 인자 내부에서는 %가 % 출력
    printf("%0+10.1f%s\n", 3.25, "%");

    return 0;
}
```

- 문자열 형식 지정자 %s와 인자로 기술
  - %[전체폭].[출력문자수]s

```
사계절은 봄 여름 가을 겨울이다.
사계절은 봄 여름 가을 겨울이다.

123456789012345
       Hel
Hel
Hello!
     Hello

123456789012345
Hi, C language!
       Hi,
Hi,
   124.568
      3.25%
+0000003.3%
```

문자열의 다양한 출력

형식지정자	인자(문자열)	결과
%5.2s	"Hello!"	___He
%7.2s	"Hello!"	_____He
%-8.6s	"Hello!"	Hello!__
%-8.3s	"Hello!"	Hel_____

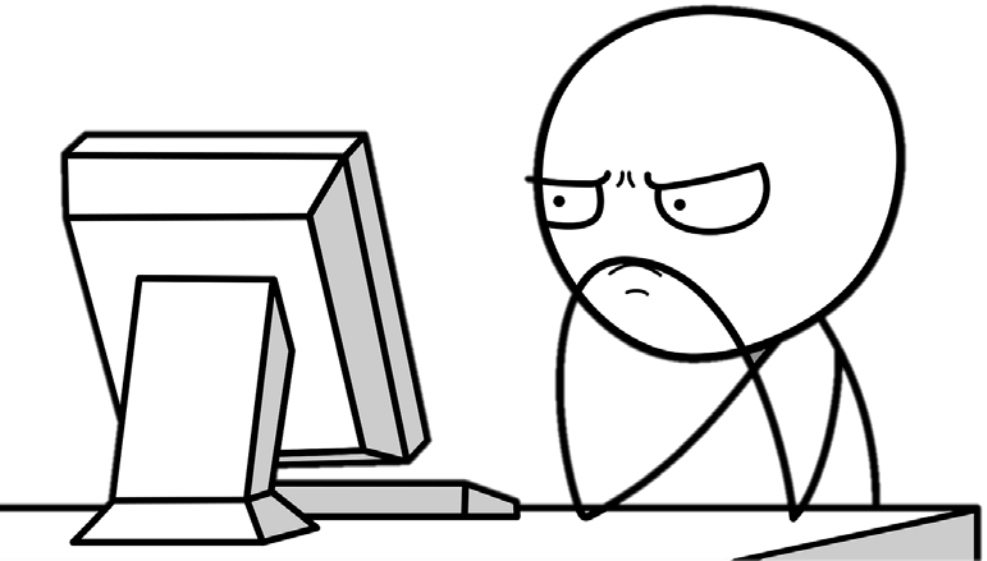
형식지정자	인자(문자열)	결과
"%5.*s", 3	"Hello!"	__Hel
"%*.4s", 7	"Hello!"	___Hell
%*.*s, 8, 6	"Hello!"	__Hello!
%*.*s, -8, 6	"Hello!"	Hello!__

Lab #02: 정수와 실수, 문자와 문자열의 출력

- 개인의 성별과 이름, 나이, 성적 등 개인 정보 출력 프로그램
  - 자료형 int 형인 나이와 double 형 성적 평균평점 등을 출력
  - 성별, 나이, 몸무게, 평균평점을 다음과 같이 출력
- 결과
  - 성별: M
  - 이름: 홍길동
  - 나이: 30
  - 몸무게: 92.489
  - 평균평점(GPA): 4.210

```c
// basicoutput.c:

#include <stdio.h>

int main(void)
{
    int age = 30;
    double gpa = 4.210f;
    char gender = 'M';
    float weight = 92.489F;

    printf("성별: [redacted]\n", [redacted]);
    printf("이름: %s\n", "[redacted]");
    printf("나이: %d\n", age);
    printf("몸무게: %.2f\n", weight);
    printf("평균평점(GPA): [redacted]\n"[redacted]);

    return 0;
}
```

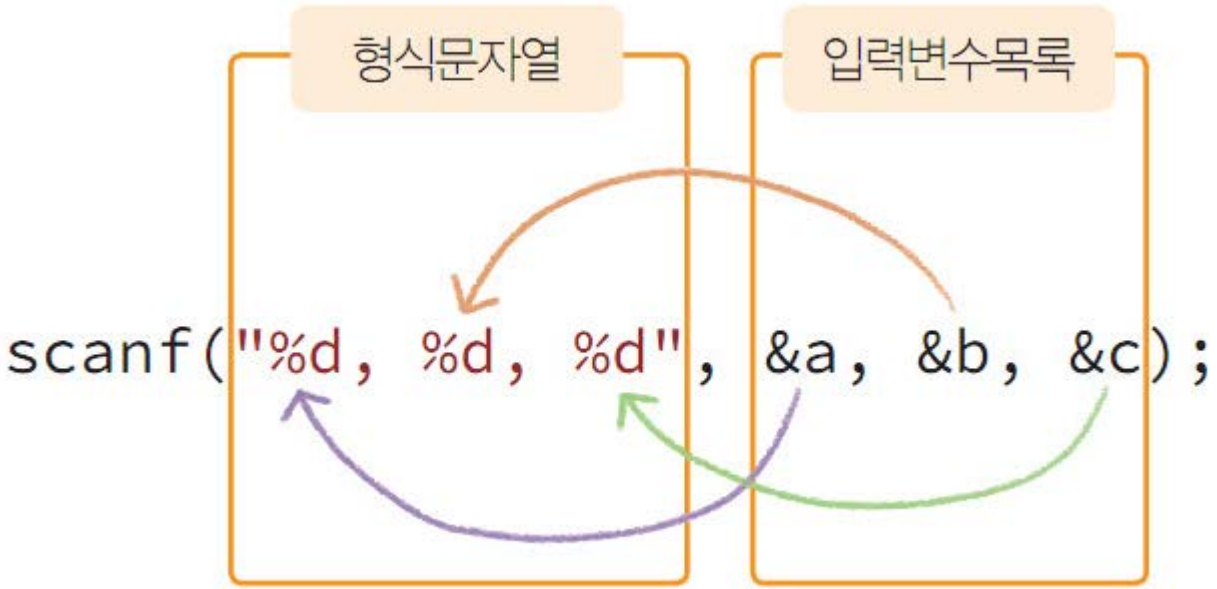
3. 입력함수 scanf()



함수 scanf()

- 대표적인 입력함수이고 %s와 %d같은 동일한 형식 지정자를 사용
  - 'scan'이라는 단어는 스캐너와 같이 어떠한 자료를 훑어 복사하거나, 유심히 살펴본다는 의미
- 표준 입력으로부터 여러 종류의 자료값을 훑어 주소연산자 &가 붙은 변수 목록에 저장
- 첫 번째 인자는 형식문자열format string
  - 형식지정자format specification는 %d, %c, %lf, %f와 같이 %로 시작
- 두 번째 인자부터는 키보드 입력값이 복사 저장되는 입력변수 목록
  - 변수이름 앞에 반드시 주소연산자 &를 붙여 나열
- 반환 유형은 int로
  - 표준입력으로 변수에 저장된 입력 개수를 반환, 다음 문장의 반환값은 3
- scanf("%d", &year)
  - 주소연산식인 &year로 사용
  - year로 기술하면 입력값이 저장될 주소를 찾지 못해 오류가 발생

함수 scanf()



Source Code #04: scanf.c

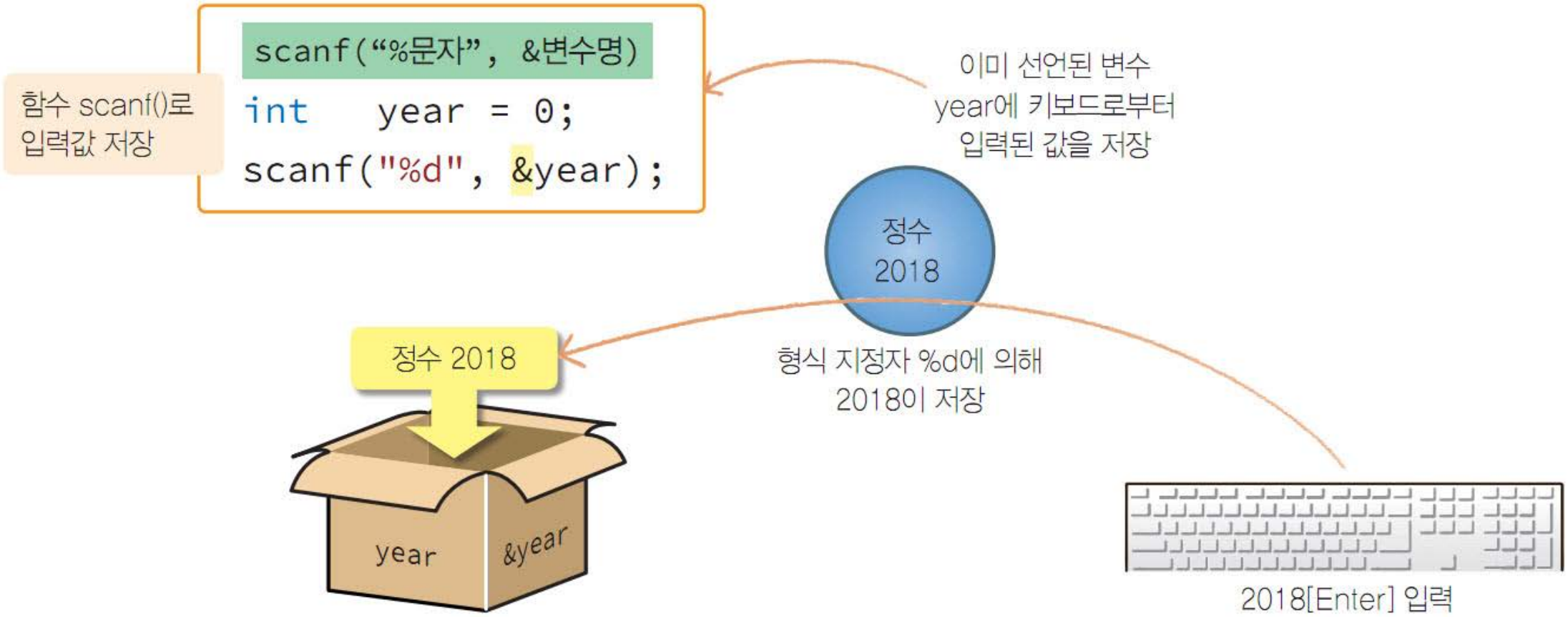
```c
// file: scanf.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
    int year = 0;
    printf("당신의 입학년도는? ");
    scanf("%d", &year); //scanf("당신의 입학년도는 ? %d", &year); 문제 발생
    printf("입학년도: %d\n\n", year);

    int month, day;
    printf("당신의 생년월일은? ");
    scanf("%d - %d - %d", &year, &month, &day);
    printf("생년월일: %d-%d-%d\n", year, month, day);

    return 0;
}
```

```
당신의 입학년도는? 2018
입학년도: 2018

당신의 생년월일은? 1990-01-01
생년월일: 1990-1-1
```

- 지정된 형식지정자에 맞게 키보드로 적당한 값을 입력한 후 [Enter] 키를 누르기 전까지는 실행을 멈춰 사용자의 입력을 기다림
- 만일 년, 월, 일을 2018-6-29과 같이 중간에 –를 넣어 입력 받으려면 함수 scanf("%d - %d - %d", …) 처럼 형식문자열에 입력 형식을 명시
- 여러 입력값을 구분해주는 구분자 separator
  - -, /, 콤마(,) 등을 사용할 수 있는데, 입력된 구분자는 형식만 체크하고 저장하지 않음

정수 입력

- 지정된 형식지정자에 맞게 키보드로 적당한 값을 입력한 후 [Enter] 키를 누르기 전까지는 실행을 멈춰 사용자의 입력을 기다림



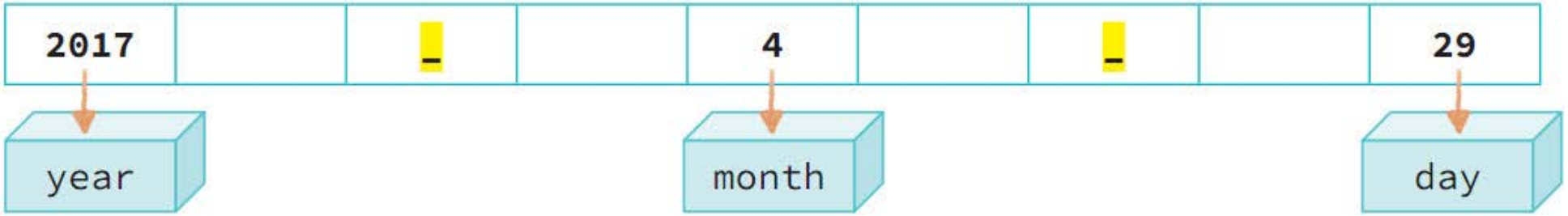
정수 입력

- 만일 년, 월, 일을 2017-4-29과 같이 중간에 –를 넣어 입력 받으려면 함수 scanf("%d - %d - %d", …) 처럼 형식문자열에 입력 형식을 명시
- 여러 입력값을 구분해주는 구분자separator
  - -, /, 콤마(,) 등을 사용할 수 있는데, 입력된 구분자는 형식만 체크하고 저장하지 않음

수를 입력 받는 경우, 형식문자열에서 빈 공간(space)은 아무 의미가 없으므로 빼도 상관없다. 실제 실행 시, 콘솔에서 년과 월 사이, 월과 일 사이에 빈 공간은 상관없이 –만 입력하면 된다.

```
scanf("%d - %d - %d", &year, &month, &day);
```

구분자인 – 반드시 필요



제어문자 %f와 %lf, %c

- 입력자료를 실수 float형 변수에 저장: 형식 지정자 %f를 사용
  - 실수 double형 변수에 저장: 형식 지정자 %lf를 사용
- 입력 자료를 문자 char형 변수에 저장: 제어문자 %c를 사용
- 출력 printf()에서 실수의 출력을 위한 형식 지정자
  - %f와 %lf를 모두 사용 가능

버퍼buffer

- 함수 scanf()는 입력에 임시저장 장소
- [Enter] 키가 원하지 않는 문자변수에 저장되어 원래 의도한 문자는 입력에 성공 못하는 일이 발생
- '버퍼'
  - 입력과 출력과 같은 자료의 흐름에서 바로 처리하지 않고 중간에 임시로 사용하는 저장 공간
  - 입력이나 출력을 바로 수행하지않고 버퍼에 저장하다가 버퍼가 모두 차거나 특정한 명령에 의해 버퍼에 있는 내용을 입력 또는 출력

Source Code #05: floatcharscan.c

```c
// file: floatcharscan.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
    float pi;
    printf("원주율을 입력하세요.\n");
    scanf("%f", &pi);
    printf("%f\n", pi);

    char ch1, ch2;
    printf("구분자를 공백으로 두 문자를 입력하세요.\n");
    //가장 앞에 공백을 두어 enter를 제거, 구분자로 공백(여러 개도 가능)을 사용
    scanf(" %c %c", &ch1, &ch2);
    printf("ch1=%c ch2=%c\n", ch1, ch2);

    return 0;
}
```

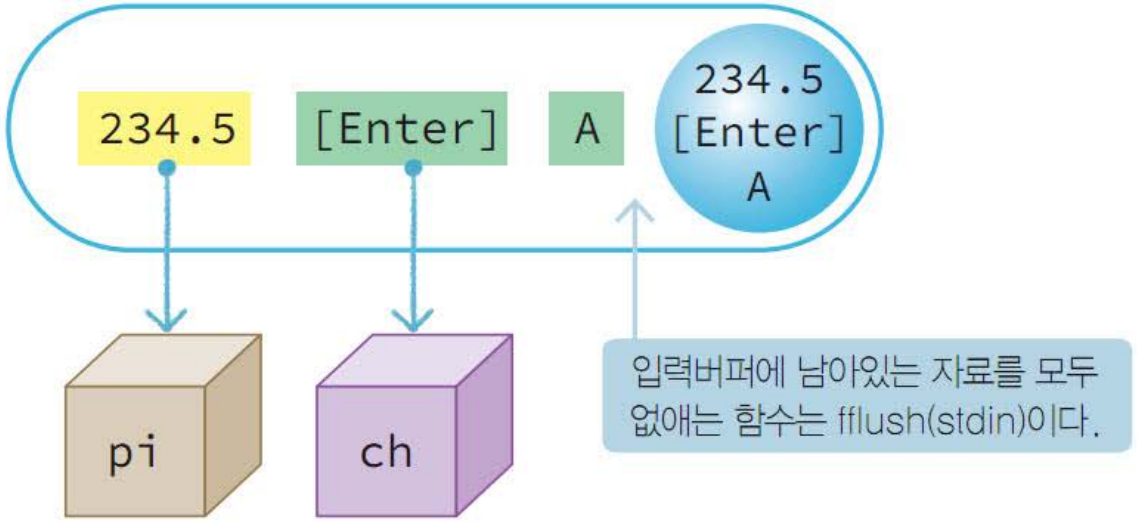
```
원주율을 입력하세요.
3.14159265
3.141593
구분자를 공백으로 두 문자를 입력하세요.
A B
ch1=A ch2=B
```

실수와 문자 입력

- 두 번의 scanf() 호출로, 콘솔에서 실수 234.5하나를 입력한 후 [Enter] 키를 누르고 다음 줄에 문자 'A'를 입력하여 변수 ch에 저장
  - 입력버퍼에는 [Enter] 키가 남아 있어, 두 번째 scanf()에서 char형 변수 ch에는 순서대로 [Enter]인 문자 '\n'가 저장되고, 실제 문자 'A'는 저장되지 않는 문제가 발생
- 이러한 문제를 해결하는 방법은 두 가지
  - 첫 번째는 버퍼에 남아있는 [Enter] 키를 함수 fflush(stdin)를 호출하여 없애버리는 방법
  - 두 번째는 문자를 입력 받는 형식지정자 %c 앞에 "공백문자를 넣어" 형식문자열을 " %c"로 지정
  - 아직 입력버퍼에 남아있는 [Enter] 키가 %c 앞에 공백문자로 인식되어 무시되고,
  - 이어 커서 위치에 입력되는 'A'가 변수 ch에 저장

실수와 문자 입력

```
scanf("%f", &pi);
...
scanf("%c", &ch);
```



해결방법 1

```
scanf("%f", &pi);

fflush(stdin); //입력 버퍼를 모두 비움

scanf("%c", &ch);
```

해결방법 2

```
scanf("%f ", &pi);

scanf(" %c", &ch);
```

Source Code #06: radixscan.c

```c
// file: radixscan.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
    int a, b, c;
    printf("십진수, 팔진수, 십육진수를 각각 입력하세요.\n");
    scanf("%d %o %x", &a, &b, &c);
    printf("%d %#o %#x\n\n", a, b, c);

    printf("십진수, 팔진수(0리딩 표현), 십육진수(0x리딩 표현)를 각각 입력하세요.\n");
    scanf("%i %i %i", &a, &b, &c);
    printf("%d %d %d\n", a, b, c);

    return 0;
}
```

```
십진수, 팔진수, 십육진수를 각각 입력하세요.
82 67 1F
82 067 0x1f

십진수, 팔진수(0리딩 표현), 십육진수(0x리딩 표현)를 각각 입력하세요.
82 067 0x1f
82 55 31
```

다양한 형식지정자

- 함수 scanf()에서 정수의 콘솔입력값
- 8진수로 인지하려면 %0를 사용
- %x는 16진수로 인지
- 다음은 함수 scanf()에서 이용되는 다양한 형식지정자

함수 scanf()의 형식 지정자

형식 지정자	콘솔 입력값의 형태	입력 변수 인자 유형
%d	십진수로 인식	정수형 int 변수에 입력값 저장
%i	십진수로 인식하며, 단 입력값에 0이 앞에 붙으면 8진수로 0x가 붙으면 16진수로 인식하여 저장	정수형 int 변수에 입력값 저장
%u	unsigned int로 인식	정수형 unsigned int 변수에 입력값 저장
%o	8진수로 인식	정수형 int 변수에 입력값 저장
%x, %X	16진수로 인식	정수형 int 변수에 입력값 저장
%f	부동소수로 인식	부동소수형 float 변수에 입력값 저장
%lf	부동소수로 인식	부동소수형 double 변수에 입력값 저장
%e, %E	지수 형태의 부동소수로 인식	부동소수형 float 변수에 입력값 저장
%c	문자로 인식	문자형 char 변수에 입력값 저장
%s	일련의 문자인 문자열(string)로 인식	문자열를 저장할 배열에 입력값 저장
%p	주소(address) 값으로 인식	정수형 unsigned int 변수에 입력값 저장

Source Code #07: putchar.c

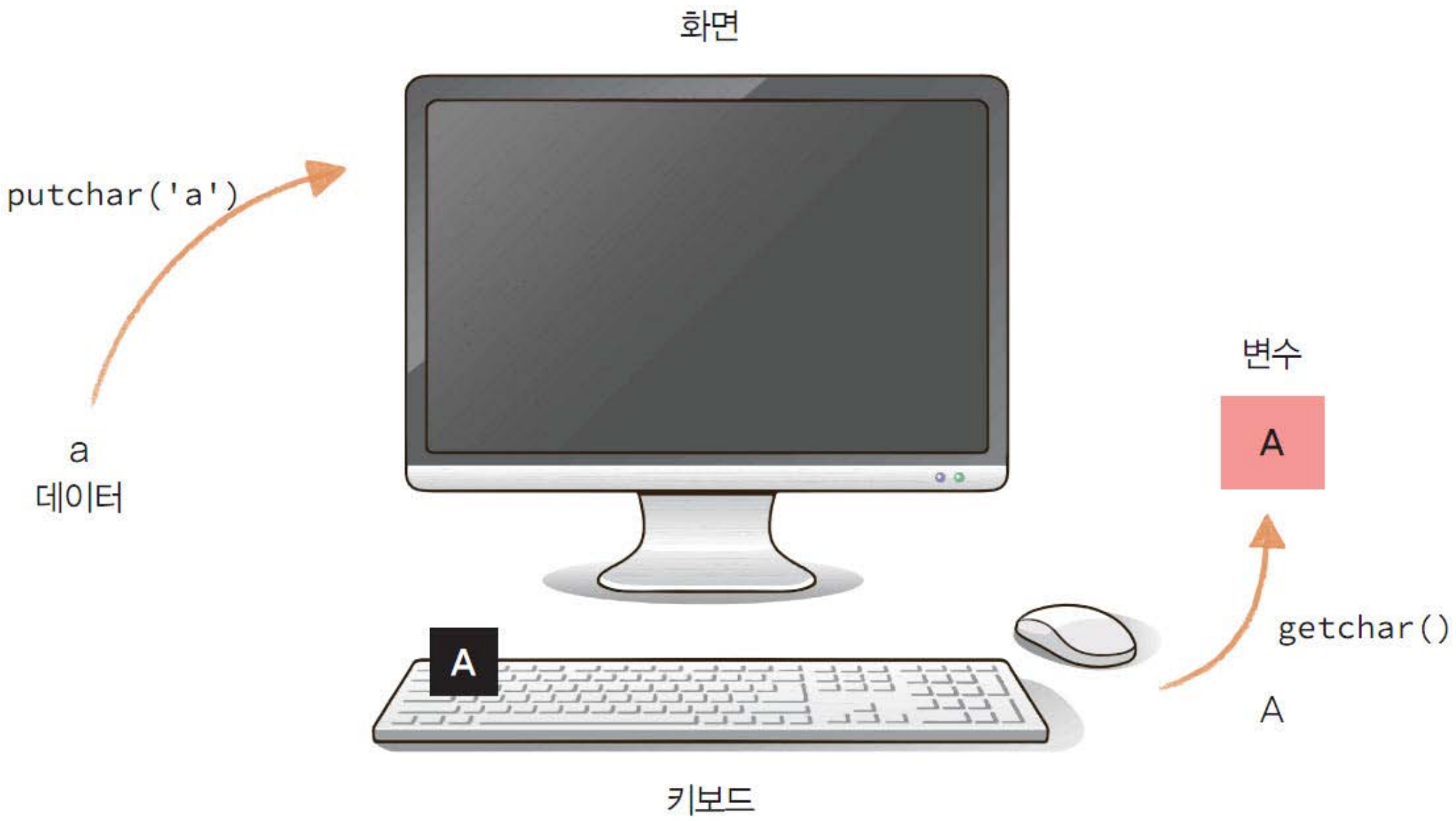
```c
// file: putchar.c

#include <stdio.h>

int main(void)
{
    char a = '\0';

    puts("문자 하나 입력:");
    a = getchar();
    putchar(a); putchar('\n');

    return 0;
}
```

```
문자 하나 입력:
#
#
```

함수 getchar()와 putchar()

- 문자의 입출력 함수
  - 함수 getchar()는 영문 'get character'의 의미로 문자 하나를 입력하는 매크로 함수
  - putchar()는 'put character'로 반대로 출력하기 위한 매크로 함수
  - 이 함수를 이용하려면 헤더파일 stdio.h 가 필요
- 함수 getchar()는 인자 없이 함수를 호출
  - 입력된 문자값을 자료형 char나 정수형으로 선언된 변수에 저장
  - char a = getchar();
- 함수호출 putchar('a')
  - 인자인 'a'를 출력하는 함수로 사용

함수 getchar()와 putchar()



Lab #03: 십진수, 팔진수, 십육진수인 세 정수를 입력 받아 적절히 출력

- 십진수, 팔진수, 십육진수인 세 정수를 입력 받아 다음 조건을 만족하도록 적절히 출력되는 프로그램
  - 세 정수를 '십진수 – 팔진수 – 십육진수'의 형식으로 입력
  - 입력과 출력
    - 세 개의 정수를 각각 다음과 같이 입력하세요. 십진수 - 팔진수 - 십육진수
    - 100 - 65 - f3
    - 입력한 수는 다음과 같습니다.
    - 100 - 65 - f3
    - 100 - 53 - 243

```c
// file: basicio.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의

#include <stdio.h>

int main(void)
{
    int dec = 30, oct = 012, hex = 0x5E;
    printf("세 개의 정수를 각각 다음과 같이 입력하세요. ");
    printf("십진수 - 팔진수 - 십육진수\n");

    scanf("%d - %o - %x", [redacted]);
    printf("\n입력한 수는 다음과 같습니다.\n");
    printf("[redacted]\n", dec, oct, hex);
    printf("[redacted]\n", dec, oct, hex);

    return 0;
}
```


I
PROGRAMMING