

C 언어 EXPRESS(개정3판)



CH06 함수와 변수



이번 장에서 학습할 내용

- 모듈화
- 함수의 개념, 역할
- 함수 작성 방법
- 반환값
- 인수 전달
- 함수를 사용하는 이유

규모가 큰 프로그램은 전체 문제를 보다 단순하고 이해하기 쉬운 함수로 나누어서 프로그램을 작성하여야 한다.

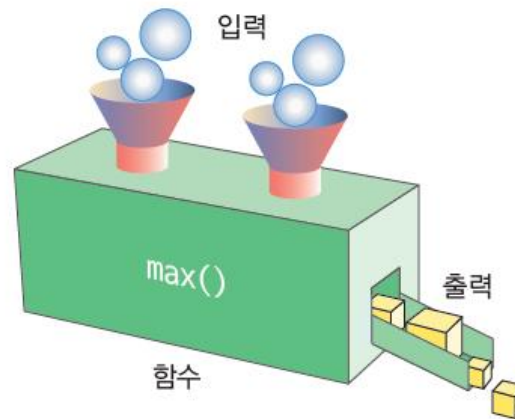




함수의 개념

● 함수란 ?

- 특정 작업을 수행하는 코드의 집합을 의미하며, 반복적으로 사용되는 코드를 한 번 정의하고 여러 곳에서 호출할 수 있도록 만든 구조
- 코드의 재사용성이 높아지고, 코드의 가독성과 유지보수성이 향상됨





예제

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

ch06_ex0.c

```
int main(void)
{
    int i;
    for (i = 0; i < 10; i++) {
        printf("*");
    }
    printf("\nHello\n");
    printf("Welcome to my world\n");
    for (i = 0; i < 10; i++) {
        printf("*");
    }
    printf("\n\n");

    return 0;
}
```

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

ch06_ex1.c

```
void print_star_world()
{
    int i;
    for(i = 0; i < 10; i++){
        printf("*");
    }
    printf("\nHello\n");
    printf("Welcome to my world\n");
    for (i = 0; i < 10; i++) {
        printf("*");
    }
    printf("\n\n");
}
```

```
int main(void)
{
    print_star_world();
    print_star_world();
    print_star_world();
    return 0;
}
```

함수 호출



함수의 종류

함수

사용자 정의 함수

라이브러리 함수



내가 원하는 함수가 없으니
직접 만들어야지!



이 함수들은 기본적으로
제공됩니다.



함수의 정의

Syntax: 함수 정의

반환형

함수 이름

예

```
void print_stars()
```

매개 변수(현재는 없다)

```
{
```

```
    for(int i=0; i<30; i++)  
        printf("*");
```

함수 몸체

```
}
```



함수 정의

- 반환형
 - 반환형은 함수가 처리를 종료한 후에 호출한 곳으로 반환하는 데이터의 유형을 말한다.
- 함수 이름
 - 함수 이름은 식별자에 대한 규칙만 따른다면 어떤 이름이라도 가능하다.
 - 함수의 기능을 암시하는 (동사+명사)를 사용하면 좋다.

```
int square()  
double compute_average()  
void set_cursor_type()
```

// 정수를 제공하는 함수
// 평균을 구하는 함수
// 커서의 타입을 설정하는 함수

함수 이름

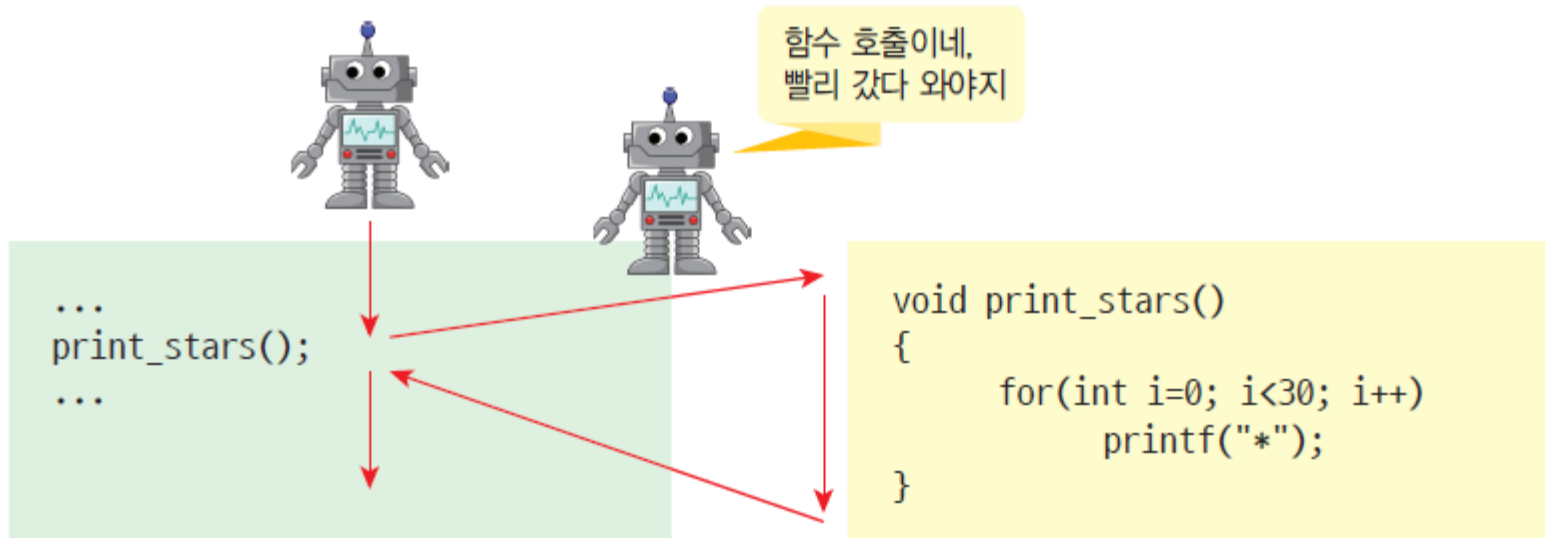


함수 호출

- 함수를 사용하려면 함수를 호출(call)하여야 한다.
- 함수 호출(function call)이란 `print_stars()`와 같이 함수의 이름을 써주는 것이다.
- 함수 안의 문장들은 호출되기 전까지는 전혀 실행되지 않는다. 함수를 호출하게 되면 현재 실행하고 있는 코드는 잠시 중단되고, 호출된 함수로 이동하여서 함수 몸체 안의 문장들이 순차적으로 실행된다.
- 호출된 함수의 실행이 끝나면 호출한 위치로 되돌아가서 잠시 중단되었던 코드가 실행을 재개한다.



함수 호출





함수는 여러 번 호출될 수 있다.



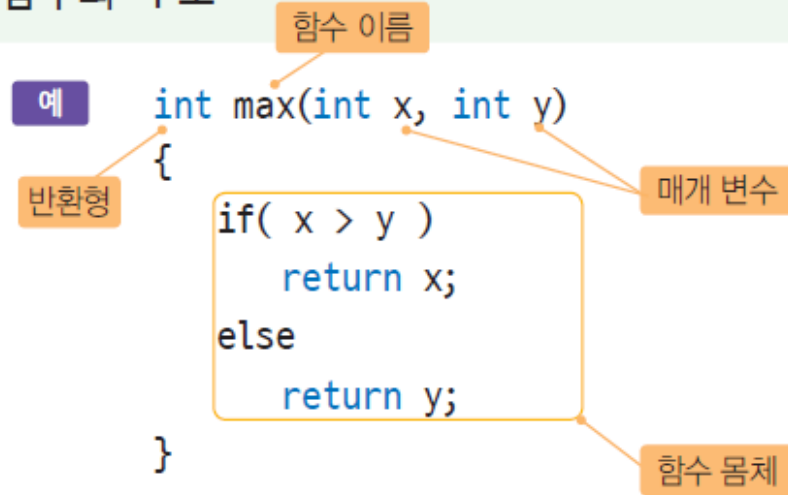
```
...  
print_stars();  
...  
print_stars();
```

```
void print_stars()  
{  
    for(int i=0; i<30; i++)  
        printf("*");  
}
```



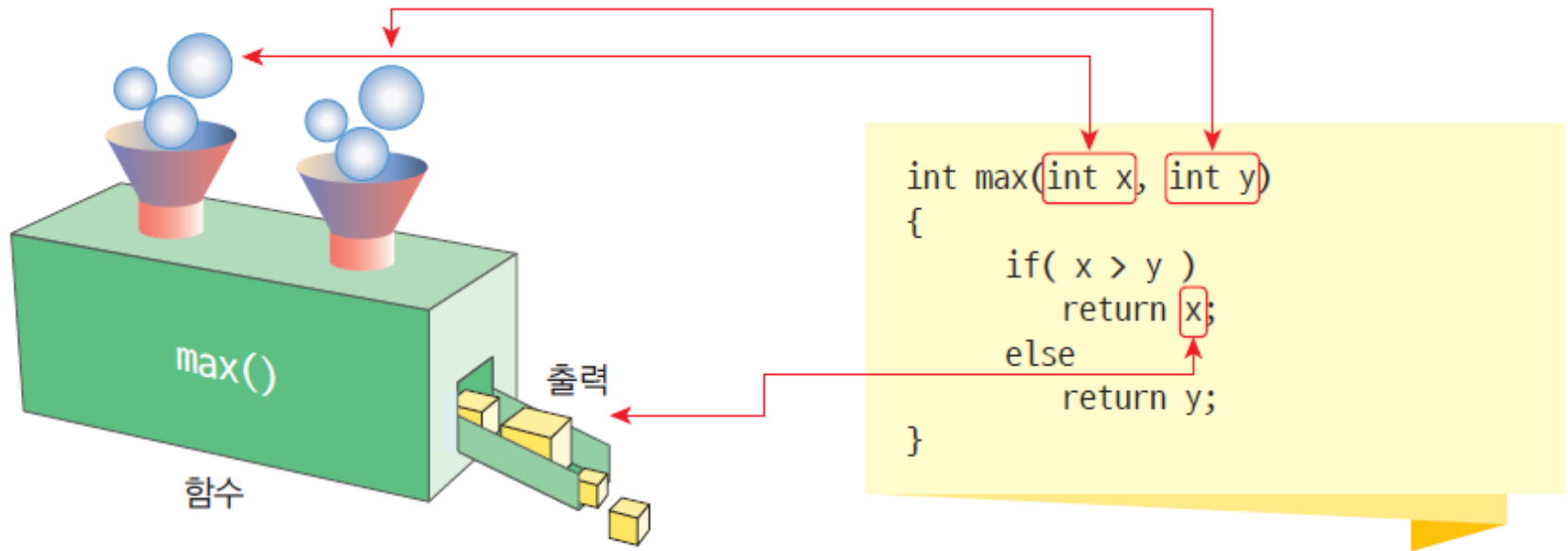
매개 변수와 반환값

Syntax: 함수의 구조





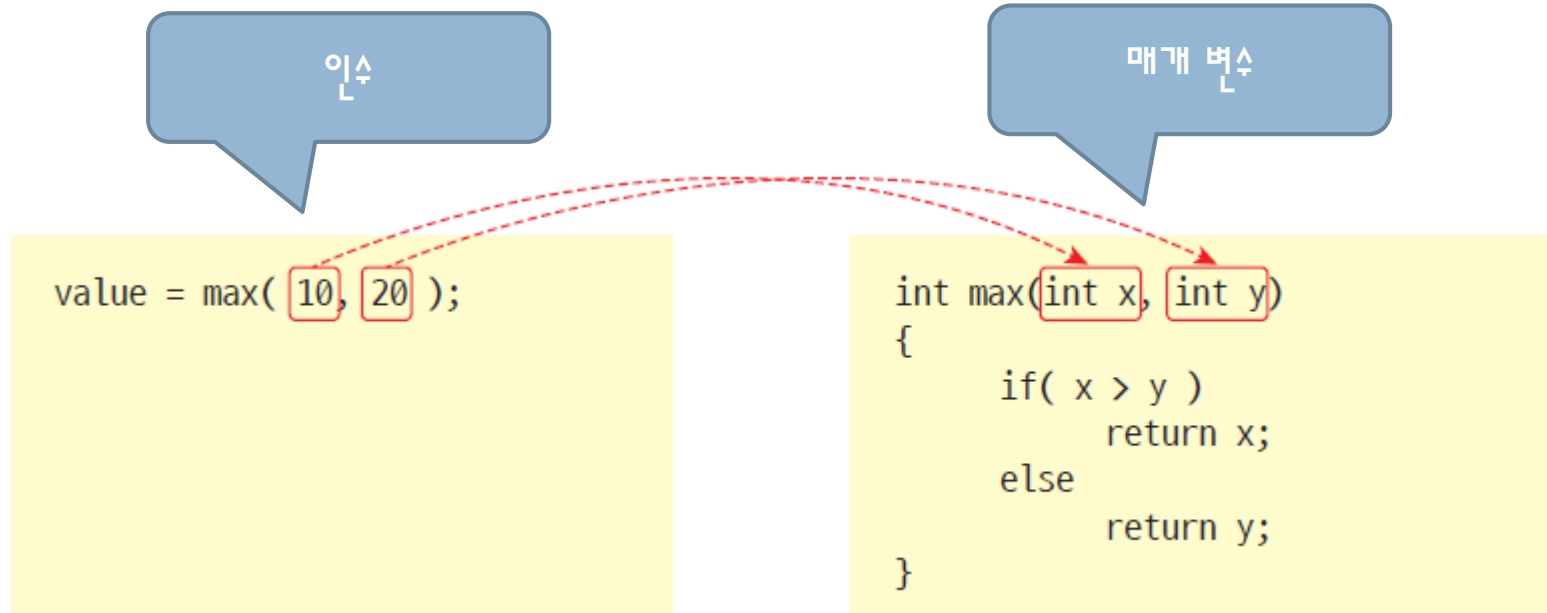
매개 변수와 반환값





인수와 매개변수

- 인수(argument)는 호출 프로그램에 의하여 함수에 실제로 전달되는 값이다.
- 매개 변수(parameter)는 이 값을 전달받는 변수이다.





인수와 매개 변수

- 만약 매개 변수가 없는 경우에는 `print_stars(void)`와 같이 매개변수 위치에 `void`를 써주거나 `print_stars()`와 같이 아무 것도 적지 않으면 된다.
- 함수가 호출될 때마다 인수는 달라질 수 있다.
- 매개 변수의 개수는 정확히 일치하여야 한다는 점이다. 매개 변수의 개수와 인수의 개수가 일치하지 않으면 아주 찾기 어려운 오류가 발생하게 된다.

```
max(10);      // max() 함수를 호출할 때는 인수가 두개이어야 한다.  
max( );      // max() 함수를 호출할 때는 인수가 두개이어야 한다.
```



반환값

- 반환값(return value)은 함수가 호출한 곳으로 반환하는 작업의 결과값이다.
- 값을 반환하려면 **return** 문장 다음에 수식을 써주면 수식의 값이 반환된다.
- 인수는 여러 개가 있을 수 있으나 반환값은 하나만 가능하다.

```
value = max( 10, 20 );
```

```
int max(int x, int y)
{
    if( x > y )
        return x;
    else
        return y;
}
```



함수 예제

No retrun, no Parameter

```
#include <stdio.h>

void print_5stars()
{
    printf("*****\n");
}

int main(void)
{
    print_5stars();
    print_5stars();
    print_5stars();
    return 0;
}
```

No retrun, Parameter

```
#include <stdio.h>

void print_5stars(int n)
{
    int i = 0;
    for(i=0;i<n;i++){
        printf("*****\n");
    }
}

int main(void)
{
    int num = 4;
    print_5stars(num);

    return 0;
}
```




함수 예제

retrun, no Parameter

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int print_5stars()
{
    int i = 0;
    int n = 0;
    int sum = 0;
    scanf("%d", &n);
    for(i=0;i<n;i++){
        printf("*****\n");
    }
    sum = n * 5;
    return sum;
}

int main(void)
{
    int res = 0;
    res=print_5stars();
    printf("%d", res);
    return 0;
}
```

retrun, Parameter

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int print_5stars(int n)
{
    int i = 0;
    int sum = 0;

    for(i=0;i<n;i++){
        printf("*****\n");
    }
    sum = n * 5;
    return sum;
}

int main(void)
{
    int num = 4;
    int res = 0;
    res=print_5stars(num);
    printf("%d", res);
    return 0;
}
```



예제

ch06_ex3.c

아래 `max()` 함수를 호출하여서 사용자가 입력한 값 중에서 더 큰 값을 찾아보자.

```
#include <stdio.h>

int max(int x, int y)
{
    int n_max = 0;
    if (x > y)
        n_max = x;
    else
        n_max = y;
    return n_max;
}

int main(void)
{
    return 0;
}
```

정수를 입력하시오: 10
정수를 입력하시오: 35
더 큰값은 35입니다.



예제

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

ch06_ex3.c

```
int max(int x, int y)
```

```
{  
    int n_max = 0;  
    if (x > y)  
        n_max = x;  
    else  
        n_max = y;  
    return n_max;  
}
```

```
int main(void)
```

```
{  
    int x=1, y=15;  
    int larger;  
    larger = max(x, y);  
    printf("더 큰값은 %d입니다. \n", larger);  
    return 0;  
}
```



참고

★ 참고사항

링크 오류

소스 파일을 컴파일하여서 실행 파일을 만들다 보면 링크 오류가 발생하는 경우가 있다. 링크 오류는 컴파일러가 함수를 찾지 못했을 때 발생한다. 즉 프로그래머가 정의되지 않은 함수를 사용하였을 때 링크 오류가 발생한다. 링크 오류가 발생하면 함수의 이름이 오류 메시지로 표시된다. 따라서 오류 메시지를 보고 함수가 확실히 정의되었는지를 확인하여야 한다.

```
1>c:\users\chun\documents\visual studio 2010\projects\hello\hello\hello.c(4): warning C4013:  
'sub'이(가) 정의되지 않았습니니다. extern은 int형을 반환하는 것으로 간주합니다.
```

```
1>hello.obj : error LNK2001: _sub 외부 기호를 확인할 수 없습니다.
```

```
1>C:\Users\chun\documents\visual studio 2010\Projects\hello\Debug\hello.exe : fatal error  
LNK1120: 1개의 확인할 수 없는 외부 참조입니다.
```

```
===== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 =====
```

링크 오류



라이브러리 함수

- 라이브러리 함수(library function): 컴파일러에서 제공하는 함수
 - 표준 입출력
 - 수학 연산
 - 문자열 처리
 - 시간 처리
 - 오류 처리
 - 데이터 검색과 정렬

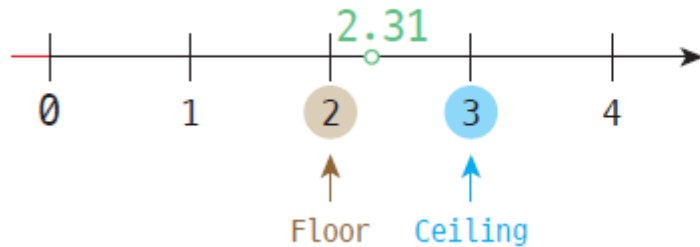




수학 라이브러리 함수

분류	함수	설명
삼각함수	<code>double sin(double x)</code>	사인값 계산
	<code>double cos(double x)</code>	코사인값 계산
	<code>double tan(double x)</code>	탄젠트값 계산
역삼각함수	<code>double acos(double x)</code>	역코사인값 계산 결과값 범위 $[0, \pi]$
	<code>double asin(double x)</code>	역사인값 계산 결과값 범위 $[-\pi/2, \pi]$
	<code>double atan(double x)</code>	역탄젠트값 계산 결과값 범위 $[-\pi/2, \pi]$
쌍곡선함수	<code>double cosh(double x)</code>	쌍곡선 코사인
	<code>double sinh(double x)</code>	쌍곡선 사인
	<code>double tanh(double x)</code>	쌍곡선 탄젠트
지수함수	<code>double exp(double x)</code>	e^x
	<code>double log(double x)</code>	$\log_e x$
	<code>double log10(double x)</code>	$\log_{10} x$
기타함수	<code>double ceil(double x)</code>	x보다 작지 않은 가장 작은 정수
	<code>double floor(double x)</code>	x보다 크지 않은 가장 큰 정수
	<code>double fabs(double x)</code>	실수 x의 절대값
	<code>int abs(int x)</code>	정수 x의 절대값
	<code>double pow(double x, double y)</code>	x^y
	<code>double sqrt(double x)</code>	\sqrt{x}

floor()와 ceil() 함수



$\lfloor x \rfloor$
floor(x)

$\lceil x \rceil$
ceil(x)

```
double result, value = 1.6;
```

```
result = floor(value);           // result는 1.0이다.  
printf("%lf ", result);
```

```
result = ceil(value);           // result는 2.0이다.  
printf("%lf ", result);
```



fabs()

```
printf("12.0의 절대값은 %f\n", fabs(12.0));  
printf("-12.0의 절대값은 %f\n", fabs(-12.0));
```




```
#define _CRT_SECURE_NO_WARNINGS                                     ch06_ex4.c
#include <stdio.h>
#include <math.h>          // 이것을 반드시 포함하여야 한다.

int main(void) {
    double result, value = 1.6;

    result = floor(value);          // result는 1.0이다.
    printf("%lf \n", result);
    result = ceil(value);           // result는 2.0이다.
    printf("%lf \n", result);

    printf("12.0의 절대값은 %f\n", fabs(12.0));
    printf("-12.0의 절대값은 %f\n", fabs(-12.0));

    return 0;
}
```

```
1.000000
2.000000
12.0의 절대값은 12.000000
-12.0의 절대값은 12.000000
```



pow()와 sqrt()

```
printf("10의 3승은 %.0f.\n", pow(10.0, 3.0));  
printf("16의 제곱근은 %.0f.\n", sqrt(64));
```

10의 3승은 1000.
16의 제곱근은 4.



cos(double x), sin(double x), tan(double x)

// 삼각 함수 라이브러리

#include <math.h>

#include <stdio.h>

int main(void)

{

double pi = 3.1415926535;

double x, y;

x = pi / 2;

y = sin(x);

printf("sin(%f) = %f\n", x, y);

y = cos(x);

printf("cos(%f) = %f\n", x, y);

printf("10의 3승은 %.0f.\n", pow(10.0, 3.0));

printf("16의 제곱근은 %.0f.\n", sqrt(64));

return 0;

}

여러 수학 함수들을 포함하는 표준
라이브러리

ch06_ex5.c

sin(1.570796) = 1.000000
cos(1.570796) = 0.000000
10의 3승은 1000.
16의 제곱근은 4.



함수를 사용하는 이유

- 소스 코드의 중복성을 없애준다.
- 한번 제작된 함수는 다른 프로그램을 제작할 때도 사용이 가능하다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.



복잡한 프로그램은 함수로 분리한다.

하나의 기능은 하나의 함수로 만든다

```
int main(void)
{
    // 숫자들의 리스트를 키보드에서 읽어들이는 코드
    ....
    // 숫자들을 크기순으로 정렬하는 코드
    ....
    // 정렬된 숫자들의 리스트를 화면에 출력하는 코드
    ...
}
```

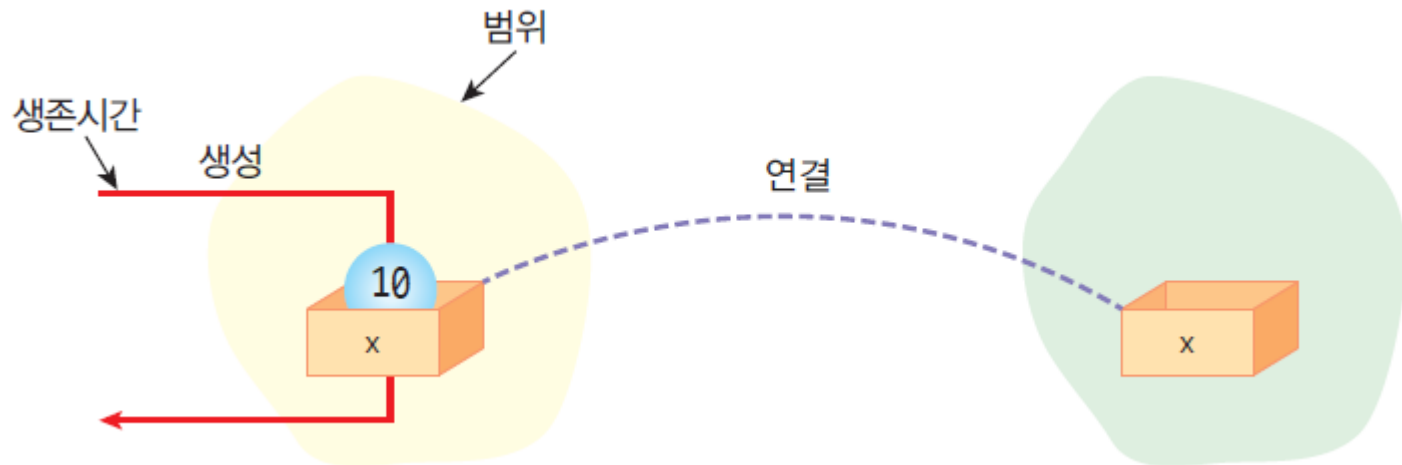


```
int main(void)
{
    ...
    read_list(); // 숫자들의 리스트를 키보드에서 읽어 들이는 함수
    sort_list(); // 숫자들의 리스트를 크기순으로 정렬하는 함수
    print_list(); // 숫자들의 리스트를 화면에 출력하는 함수
    ...
}
```



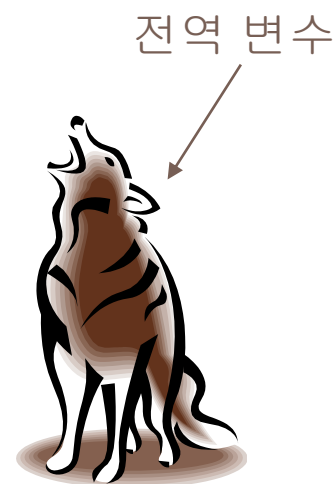
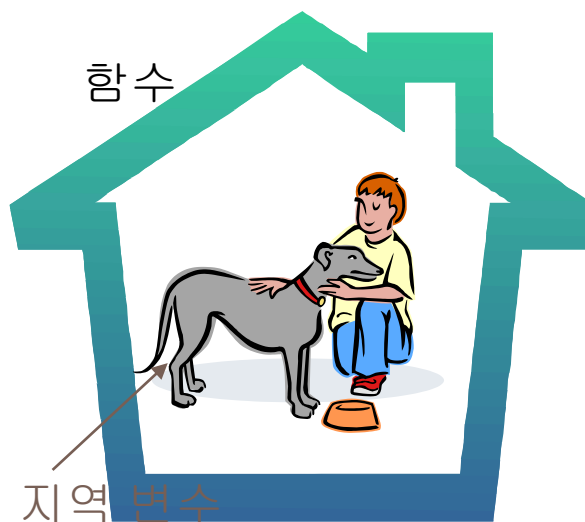
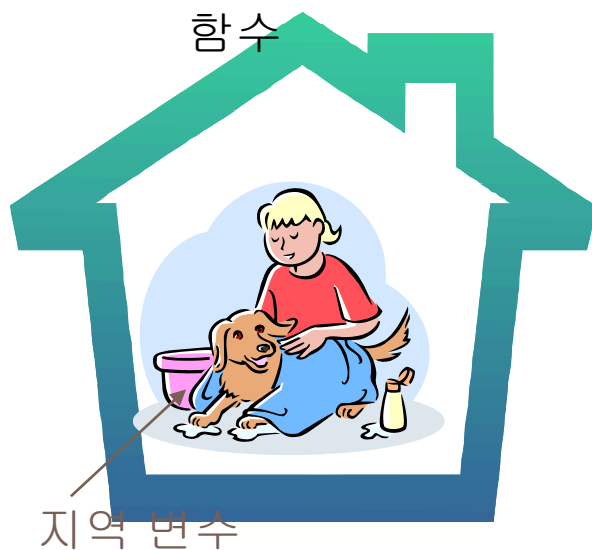
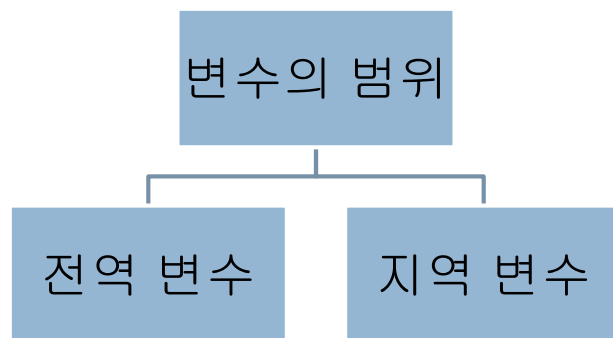
변수의 속성

- 변수의 속성 : 이름, 타입, 크기, 값 + 범위, 생존 시간, 연결
 - 범위(scope) : 변수가 사용 가능한 범위, 가시성
 - 생존 시간(lifetime) : 메모리에 존재하는 시간
 - 연결(linkage) : 다른 영역에 있는 변수와의 연결 상태



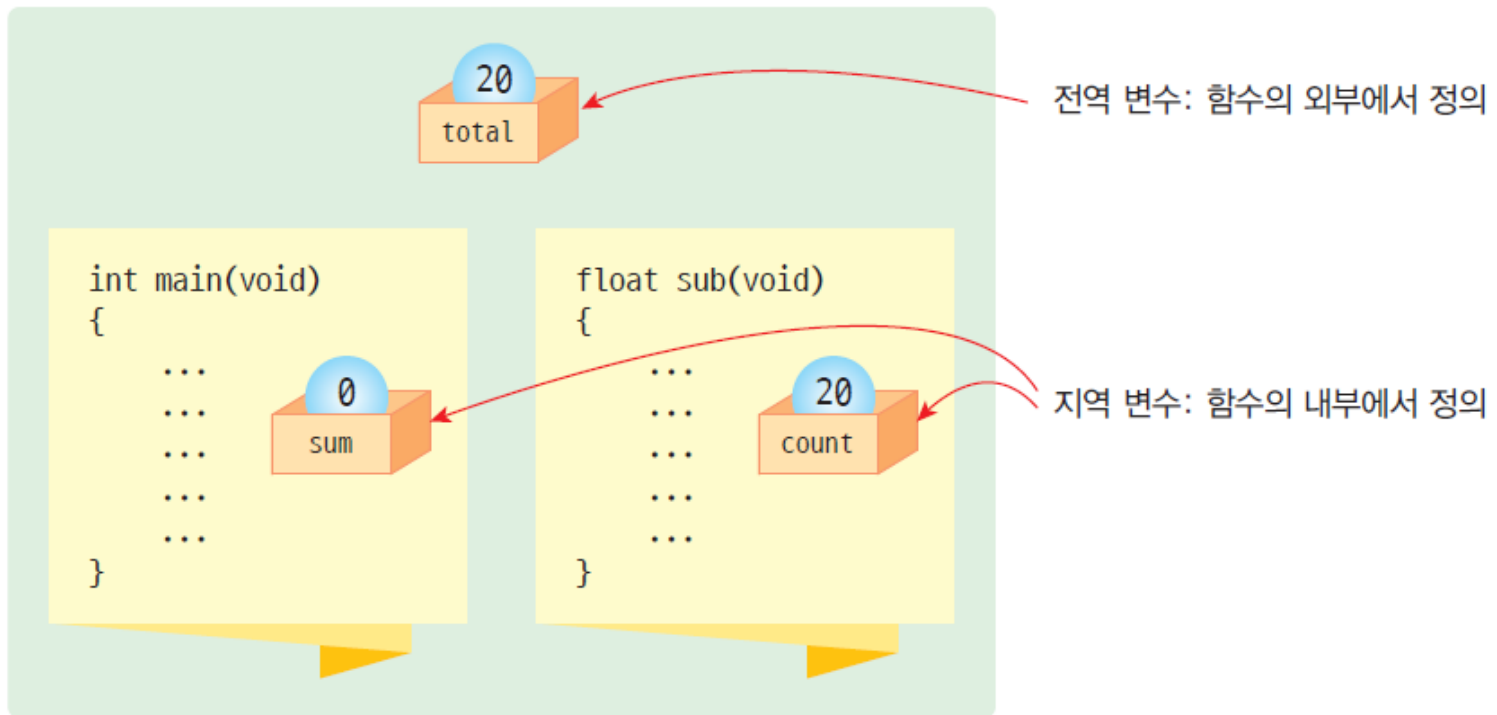


변수의 범위





전역 변수와 지역 변수





지역 변수

- 지역 변수(**local variable**)는 블록 안에 선언되는 변수

```
int sub(void)
```

```
{
```

```
    int x = 0;
```

```
    while(flag != 0){
```

```
        int y;
```

```
        ...
```

```
    }
```

```
    y = 0;    // 오류!!
```

```
    ...
```

```
}
```

지역 변수 x가 사용가능한 범위

지역 변수 y가 사용가능한 범위

지역 변수는 선언된
블록을 떠나면 안됩니다.



y가 선언된 블록을 벗어나서
사용하였으므로 오류!



지역 변수 선언 위치

- 최신 버전의 C에서는 블록 안의 어떤 위치에서도 선언 가능!!

```
while(flag!=0) {
```

```
...
```

```
int x = get_integer();
```

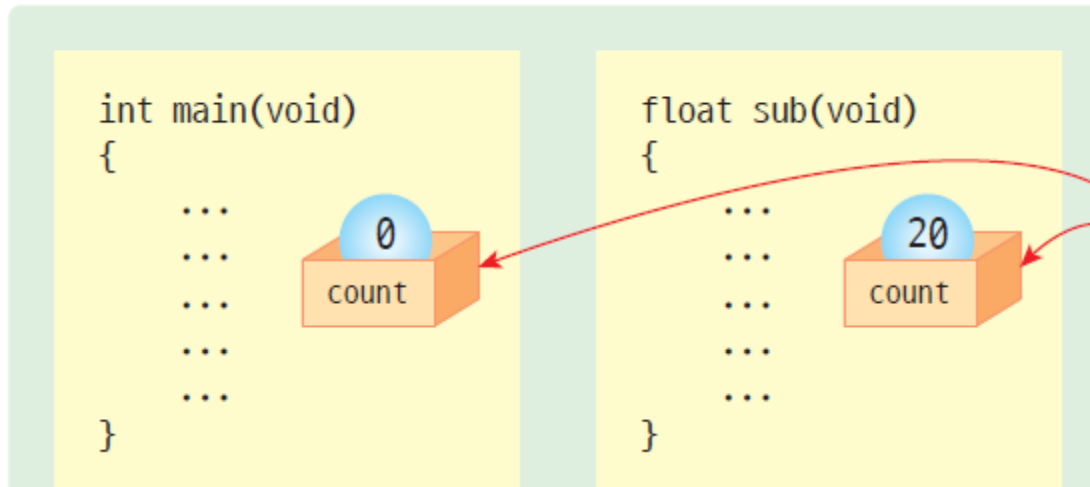
```
...
```

```
}
```

블록의 중간에서도 얼마든지
지역 변수를 선언할 수 있다.



이름이 같은 지역 변수

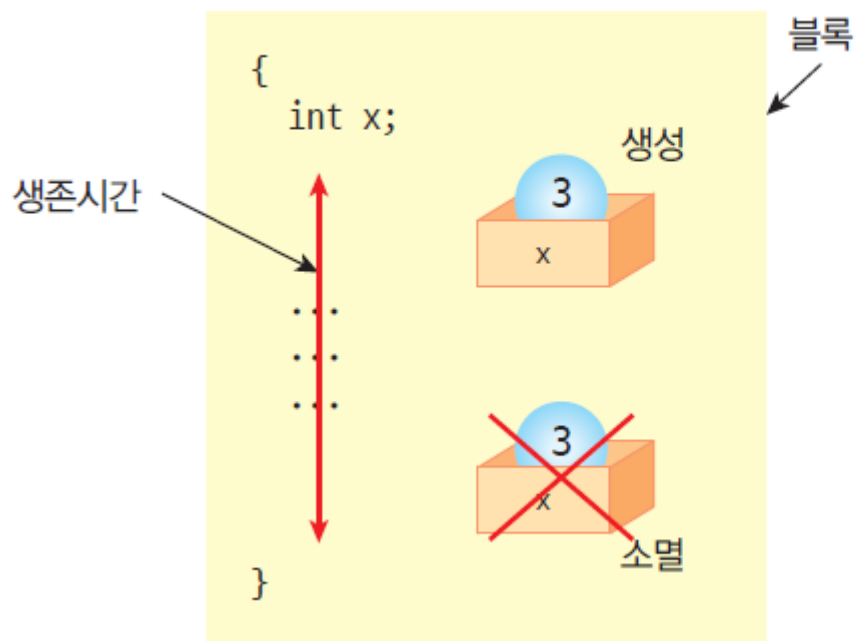


블록만 다르면
이름은 같아도 됩니다.





지역 변수의 생성 기간



지역 변수는 선언된 블록이 끝나면 자동으로 소멸됩니다.





함수의 매개 변수

- 함수의 헤더 부분에 정의되어 있는 매개 변수도 일종의 지역 변수이다. 즉 지역 변수가 지니는 모든 특징을 가지고 있다.
- 지역 변수와 다른 점은 함수 호출시의 인수 값으로 초기화되어 있다는 점이다.

```
int inc(int counter)
```

```
{
```

```
    counter++;
```

```
    return counter;
```

```
}
```

매개 변수도 일종의
지역 변수



지역 변수 예제

ch06_ex6.c

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

```
int main(void)
```

```
{
```

```
    int i=0;
```

```
    for(i = 0; i < 5; i++)
```

```
    {
```

```
        int res = 0;
```

```
        res = add(i, i + 1);
```

```
        printf("%d+%d = %d\n", i, i+1, res);
```

```
    }
```

```
    return 0;
```

```
}
```

```
int add(int x, int y)
```

```
{
```

```
    int sum = 0;
```

```
    sum = x + y;
```

```
    return sum;
```

```
}
```

지역 변수 블록이
시작할 때 마다
생성되어 초기화된다.

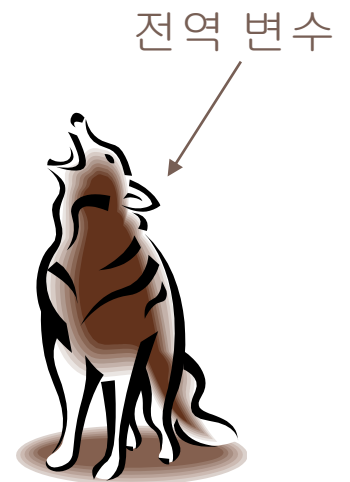
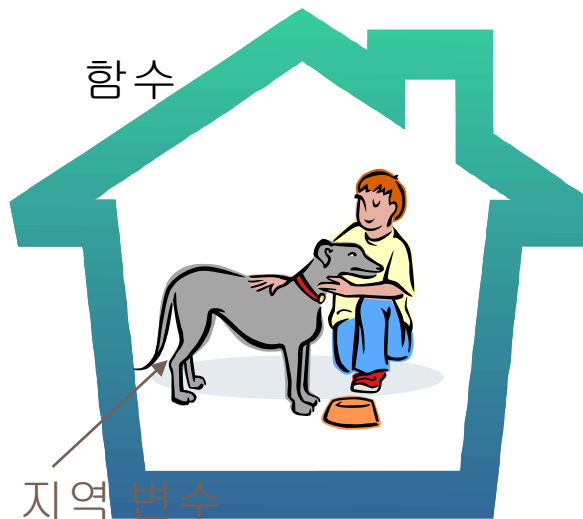
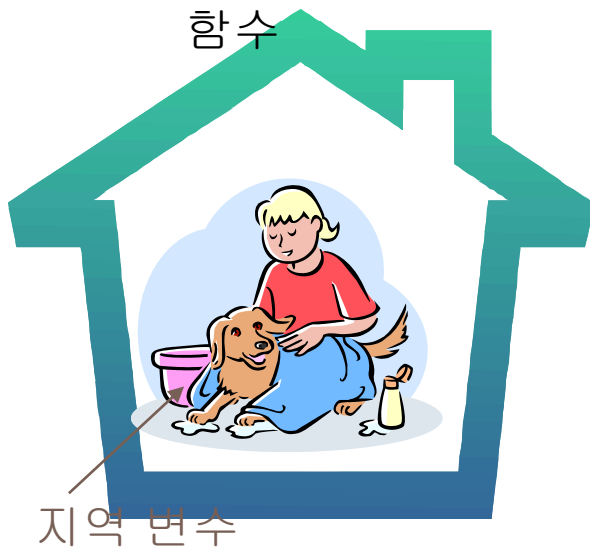
매개 변수도 일종의
지역 변수임

0+1 = 1
1+2 = 3
2+3 = 5
3+4 = 7
4+5 = 9



전역 변수

- 전역 변수(**global variable**)는 함수 외부에서 선언되는 변수이다.
- 전역 변수의 범위는 소스 파일 전체이다.





전역 변수의 초기값, 생조기간, 사용

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

ch06_ex7.c

```
int x;
```

전역 변수
초기값은 0

```
void sub();
```

```
int main(void)
```

```
{
```

```
    for (x = 0; x < 10; x++)  
        sub();
```

```
}
```

```
void sub()
```

```
{
```

```
    for (x = 0; x < 10; x++)  
        printf("*");
```

```
}
```

전역
변수의
범위



전역 변수의 사용

- 거의 모든 함수에서 사용하는 공통적인 데이터는 전역 변수로 한다.
- 일부의 함수들만 사용하는 데이터는 전역 변수로 하지 말고 함수의 인수로 전달한다.



같은 이름의 전역 변수와 지역 변수

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

ch06_ex8.c

```
int x;
```

```
void sub();
```

```
int main(void)
```

```
{
```

```
    int x = 10;
```

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

```
    for (x = 0; x < 10; x++)
```

```
        sub();
```

```
}
```

```
void sub()
```

```
{
```

```
    for (x = 0; x < 10; x++)
```

```
        printf("*");
```

```
}
```

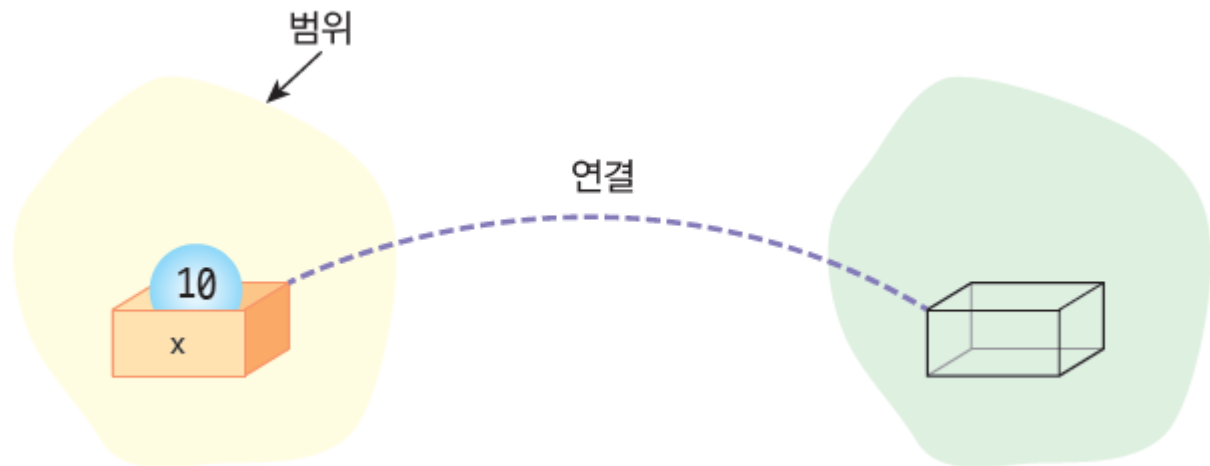
같은 변수명으로
전역, 지역 변수 선언

10



연결

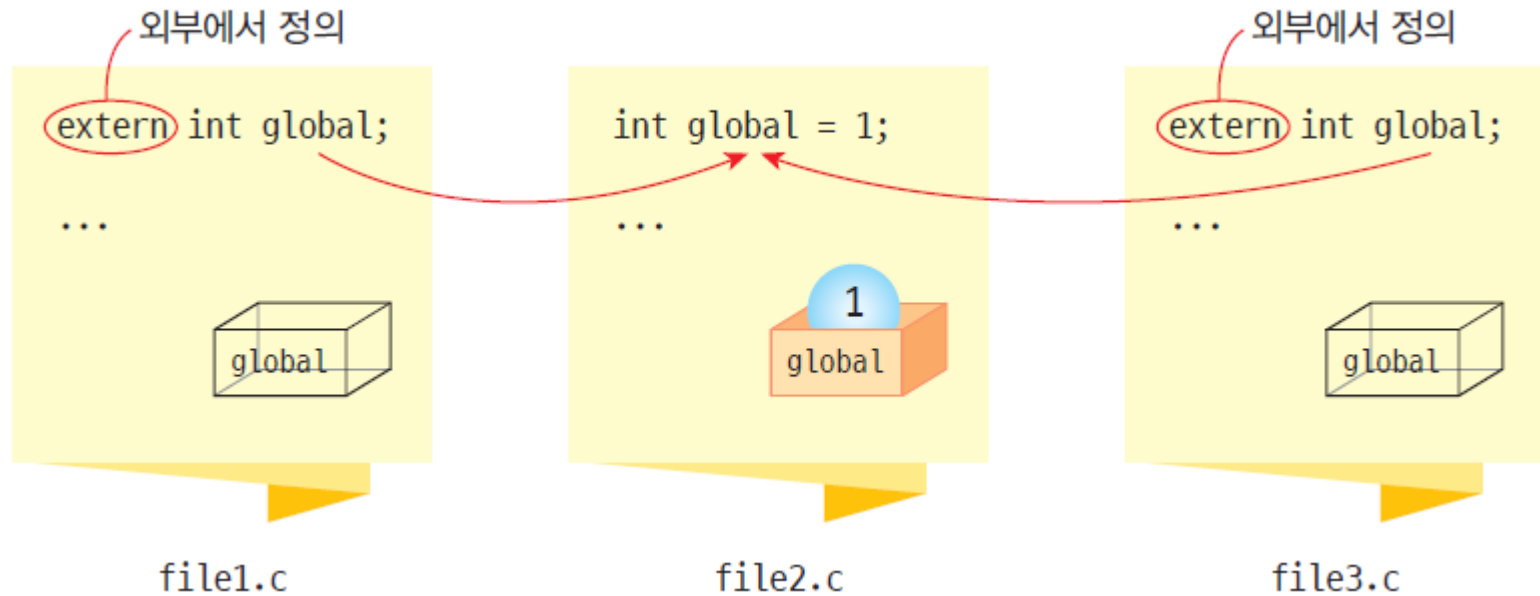
- *연결(linkage)*: 다른 범위에 속하는 변수들을 서로 연결하는 것
 - 외부 연결
 - 내부 연결
 - 무연결
- 전역 변수만이 연결을 가질 수 있다.





외부 연결

- 전역 변수를 **extern**을 이용하여서 서로 연결





```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

extern int max(int x, int y);
extern int max_np();

int n1 = 2;
int n2 = 3;

int main(void)
{
    int i = 4;
    int j = 5;
    int r_max = 0, r_max_np = 0;
    r_max = max(i, j);
    printf("%d\n", r_max);
    r_max_np = max_np();
    printf("%d\n", r_max_np);
    return 0;
}
```

```
extern int n1;
extern int n2;
int max(int x, int y) {
    int n_max = 0;
    if (x > y) {
        n_max = x;
    }
    else {
        n_max = y;
    }
    return n_max;
}

int max_np() {
    int n_max = 0;
    if (n1 > n2) {
        n_max = n1;
    }
    else {
        n_max = n2;
    }
    return n_max;
}
```



저장 유형 정리

- 일반적으로는 *자동 저장 유형* 사용 권장
- 변수의 값이 함수 호출이 끝나도 그 값을 유지하여야 할 필요가 있다면 *지역 정적*
- 만약 많은 함수에서 공유되어야 하는 변수라면 *외부 참조 변수*

저장 유형	키워드	정의되는 위치	범위	생존 시간
자동	auto	함수 내부	지역	임시
레지스터	register	함수 내부	지역	임시
정적 지역	static	함수 내부	지역	영구
전역	없음	함수 외부	모든 소스 파일	영구
정적 전역	static	함수 외부	하나의 소스 파일	영구
외부 참조	extern	함수 외부	모든 소스 파일	영구



Lab: 소수 찾기

- 주어진 숫자가 소수(prime)인지를 결정하는 프로그램이다.
- 양의 정수 n 이 소수가 되려면 1과 자기 자신만을 약수로 가져야 한다.
- 암호학에서 많이 사용

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100



Lab: 소수 찾기

정수를 입력하시오: 12229
12229은 소수가 아닙니다.



알고리즘

1. 사용자로부터 정수를 입력받아서 변수 n 에 저장한다.
2. 약수의 개수를 0으로 초기화한다.
3. `for(i=1; i<=n ; i++)`
4. n 을 i 로 나누어서 나머지가 0인지 본다.
5. 나머지가 0이면 약수의 개수를 증가한다.
6. 약수의 개수가 2이면 정수 n 은 소수이다.



```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int get_integer(void)
```

```
{
```

```
    int n;
```

```
    printf("정수를 입력하시오: ");
```

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

```
    return n;
```

```
}
```

```
int is_prime(int n)
```

```
{
```

```
    int i;
```

```
    for (i = 2; i < n; i++)
```

```
    {
```

```
        if (n % i == 0)
```

```
            return 0;
```

```
    }
```

```
    return 1;
```

```
}
```



```
int main(void)
{
    int n, result;
    n = get_integer();

    if (is_prime(n) == 1)
        printf("%d은 소수입니다.\n", n);
    else
        printf("%d은 소수가 아닙니다.\n", n);
    return 0;
}
```



도전문제

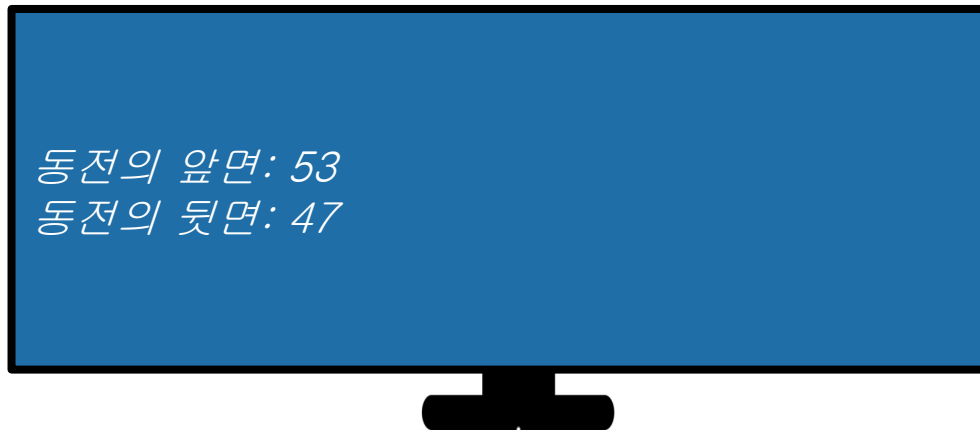
1. 1부터 사용자가 입력한 숫자 n 사이의 모든 소수를 찾도록 위의 프로그램을 변경하여 보자.





lab: 동전 던지기 게임

- 동전을 100번 던져서 앞면이 나오는 횟수와 뒷면이 나오는 횟수를 출력한다.





```
#define _CRT_SECURE_NO_WARNINGS
```

```
#include <stdlib.h>
```

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

```
#include <time.h>
```

```
int coin_toss(void);
```

```
int main(void)
```

```
{
```

```
    int toss;
```

```
    int heads = 0;
```

```
    int tails = 0;
```

```
    srand((unsigned)time(NULL));
```

```
    for (toss = 0; toss < 100; toss++) {
```

```
        if (coin_toss() == 1)
```

```
            heads++;
```

```
        else
```

```
            tails++;
```

```
    }
```



```
printf("동전의 앞면: %d \n", heads);  
printf("동전의 뒷면: %d \n", tails);  
return 0;  
  
}  
  
int coin_toss(void)  
{  
    int i = rand() % 2;  
    if (i == 0)  
        return 0;  
    else  
        return 1;  
}
```



Lab: 자동차 게임

- 난수를 이용하여서 자동차 게임을 작성

```
CAR #1:*****  
CAR #2:*****
```




알고리즘

1. 난수 발생기를 초기화한다
2. `for(i=0; i<주행시간; i++)`
3. 난수를 발생하여서 자동차1의 주행거리에 누적한다.
4. 난수를 발생하여서 자동차2의 주행거리에 누적한다.
5. `disp_car()`를 호출하여서 자동차1을 화면에 *표로 그린다.
6. `disp_car()`를 호출하여서 자동차2을 화면에 *표로 그린다.



```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void disp_car(int car_number, int distance)
{
    int i;
    printf("CAR #%d:", car_number);
    for( i = 0; i < distance/10; i++ ) {
        printf("*");
    }
    printf("\n");
}
```



```
int main(void)
{
    int i;
    int car1_dist=0, car2_dist=0;

    srand( (unsigned)time( NULL ) );

    for( i = 0; i < 6; i++ ) {
        car1_dist += rand() % 100;
        car2_dist += rand() % 100;
        disp_car(1, car1_dist);
        disp_car(2, car2_dist);
        printf("-----\n");
        _getch();
    }
    return 0;
}
```

rand()를 이용하여서 난수를 발생
다. 난수의 범위는 %연산자를 사
하여서 0에서 99로 제한하였다.



도전문제

- 자동차를 3개로 늘려보자.



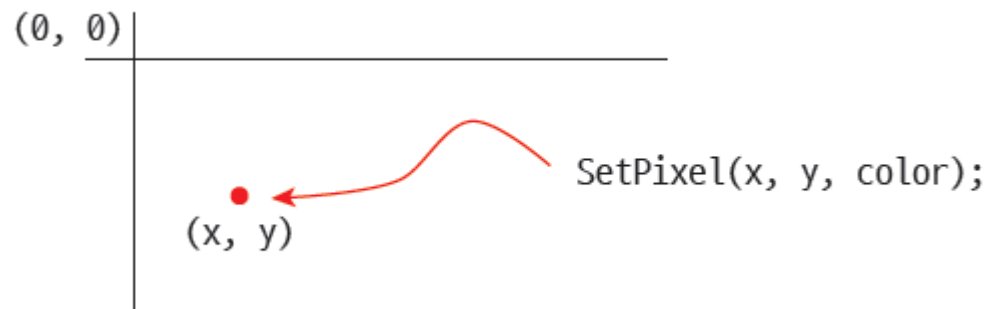


Lab: 불규칙하게 점 그리기





점을 그리는 함수



```
x = rand() % 300;           // x는 0에서 299 사이의 랜덤한 값
y = rand() % 300;           // y는 0에서 299 사이의 랜덤한 값

red = rand() % 256;          // red는 0에서 255 사이의 랜덤한 값
green = rand() % 256;        // green은 0에서 255 사이의 랜덤한 값
blue = rand() % 256;         // blue는 0에서 255 사이의 랜덤한 값

// (x, y) 위치에 (red, green, blue) 색상으로 점을 그린다.
SetPixel(hdc, x, y, RGB(red, green, blue));
```



```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <conio.h>

int main(void)
{
    int i, x, y, red, green, blue;

    HDC hdc;
    hdc = GetWindowDC(GetForegroundWindow());

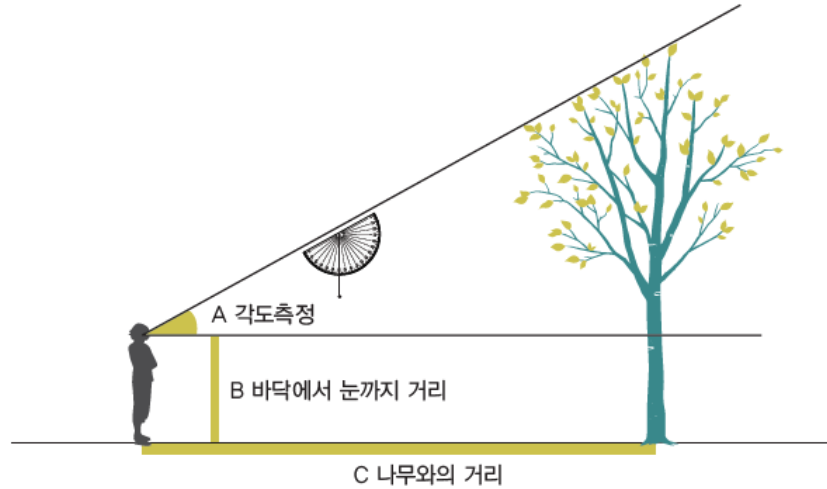
    srand((unsigned)time(NULL));
```



```
for (i = 0; i < 10000; i++)  
{  
    x = rand() % 300; // x는 0에서 299 사이의 랜덤한 값  
    y = rand() % 300; // y는 0에서 299 사이의 랜덤한 값  
  
    red = rand() % 256; // red는 0에서 255 사이의 랜덤한 값  
    green = rand() % 256; // green은 0에서 255 사이의 랜덤한 값  
    blue = rand() % 256; // blue는 0에서 255 사이의 랜덤한 값  
  
    SetPixel(hdc, x, y, RGB(red, green, blue));  
}  
_getch();  
return 0;  
}
```




Lab: 나무 높이 측정



나무와의 길이(단위는 미터): 4.2

측정자의 키(단위는 미터): 1.8

각도(단위는 도): 62

나무의 높이(단위는 미터): 9.699047



```
#define _CRT_SECURE_NO_WARNINGS
#include <math.h>
#include <stdio.h>
```

```
int main(void)
{
    double height, distance, tree_height, degrees, radians;

    printf("나무와의 길이(단위는 미터): ");
    scanf("%lf", &distance);

    printf("측정자의 키(단위는 미터): ");
    scanf("%lf", &height);

    printf("각도(단위는 도): ");
    scanf("%lf", &degrees);

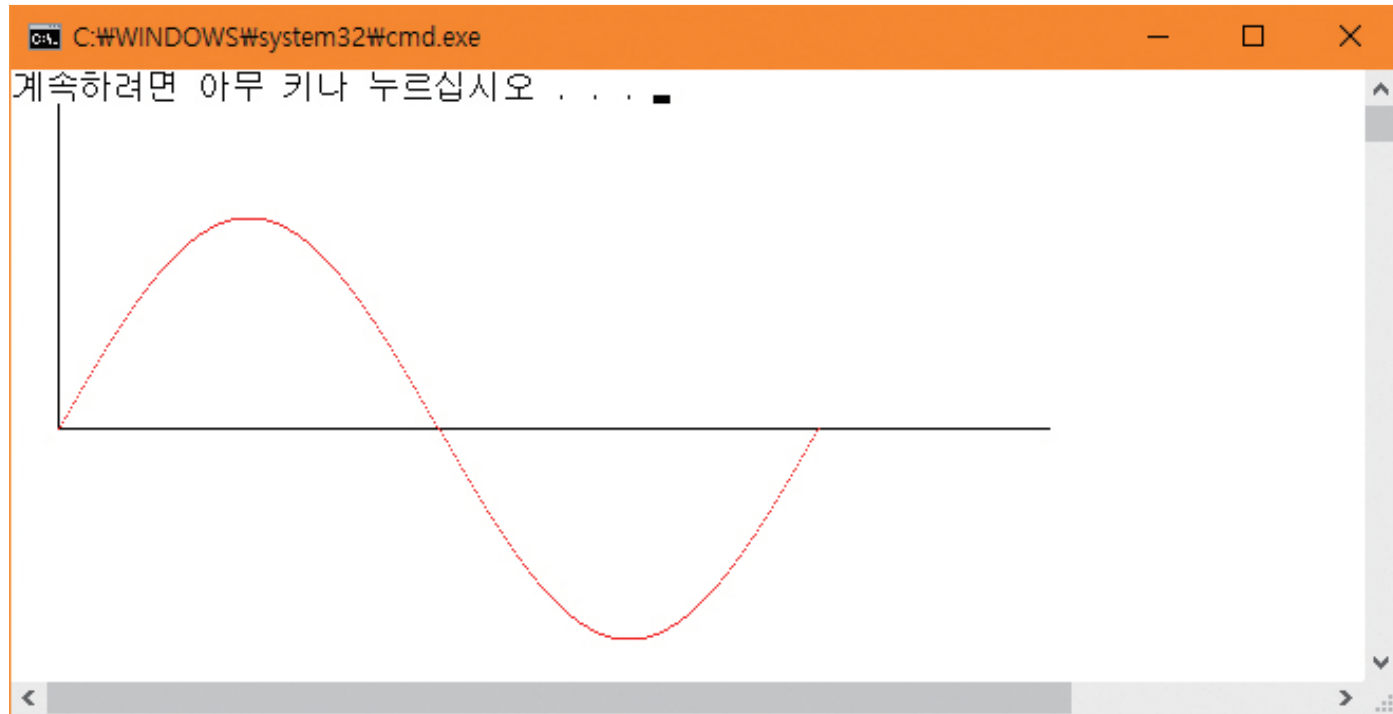
    radians = degrees * (3.141592 / 180.0);

    tree_height = tan(radians) * distance + height;
    printf("나무의 높이(단위는 미터): %lf \n", tree_height);

    return 0;
}
```



Lab: 삼각함수 그리기





```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>

#include <stdio.h>
#include <math.h>
#define PI 3.141592

double rad(double degree)
{
    return PI * degree / 180.0;
}
int main(void)
{

    int degree, x, y;
    double radian, result;

    HWND hwnd = GetForegroundWindow();
    HDC hdc = GetWindowDC(hwnd);
```



```
HWND hwnd = GetForegroundWindow();
HDC hdc = GetWindowDC(hwnd);

MoveToEx(hdc, 30, 200, 0);
LineTo(hdc, 500, 200);

MoveToEx(hdc, 30, 200, 0);
LineTo(hdc, 30, 0);

for (degree = 0; degree <= 360; degree++)
{
    result = sin(rad((double)degree));
    x = degree + 30;
    y = 200 - (int)(100.0 * result);
    SetPixel(hdc, x, y, RGB(255, 0, 0));
}
return 0;
}
```



Mini Project

- 사용자가 입력한 실수들의 합을 계산하는 프로그램을 작성해보자.

```
실수를 입력하시오: 10  
실수를 입력하시오: 20  
실수의 합=30.000000
```



Mini Project

- C 프로그램 작성 시에 도움을 줄 수 있는 함수들을 몇 가지 작성하고 테스트하자.
 - `int get_integer()`: 안내 메시지를 출력하고 정수를 받아서 반환한다.
 - `double get_double()`: 안내 메시지를 출력하고 `double`형의 실수를 받아서 반환한다.
 - `char get_char()`: 안내 메시지를 출력하고 문자를 받아서 반환한다.



```
#include <stdio.h>
int get_integer();
double get_double();
char get_char();
int main(void)
{
    double f, g;
    f = get_double();
    g = get_double();
    printf("실수의 합=%lf\n", f + g);
    return 0;
}
int get_integer() {
    int n;
    printf("정수를 입력하시오: ");
    scanf("%d", &n);
    return n;
}
```



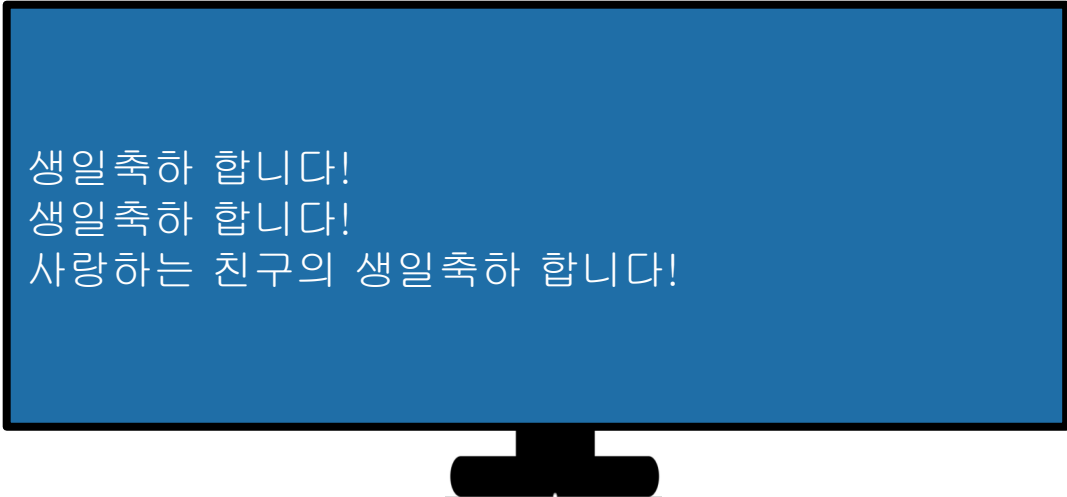

```
double get_double()
{
    double n;
    printf("실수를 입력하시오: ");
    scanf("%lf", &n);
    return n;
}

char get_char()
{
    char n;
    printf("문자를 입력하시오: ");
    scanf(" %c", &n);
    return n;
}
```



Lab: 생일 축하 함수

- 생일 축하 메시지를 출력하는 함수 `happyBirthday()`를 작성해보자.



생일 축하 합니다!
생일 축하 합니다!
사랑하는 친구의 생일 축하 합니다!



예제

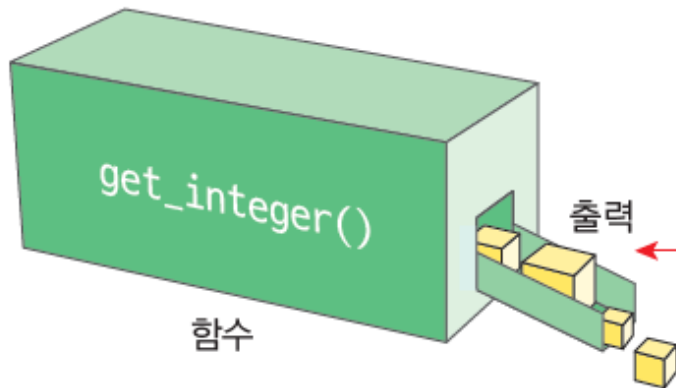
```
#include <stdio.h>

void happyBirthday()
{
    printf("생일 축하 합니다! \n");
    printf("생일 축하 합니다! \n");
    printf("사랑하는 친구의 ");
    printf("생일 축하 합니다! \n");
}

int main(void)
{
    happyBirthday();
    return 0;
}
```



Lab: 정수를 입력받는 get_integer() 함수



```
int get_integer()
{
    int value;
    printf("정수를 입력하시오: ");
    scanf("%d", &value);
    return value;
}
```



get_integer() 함수

```
// 사용자로부터 정수를 받는 함수
#include <stdio.h>

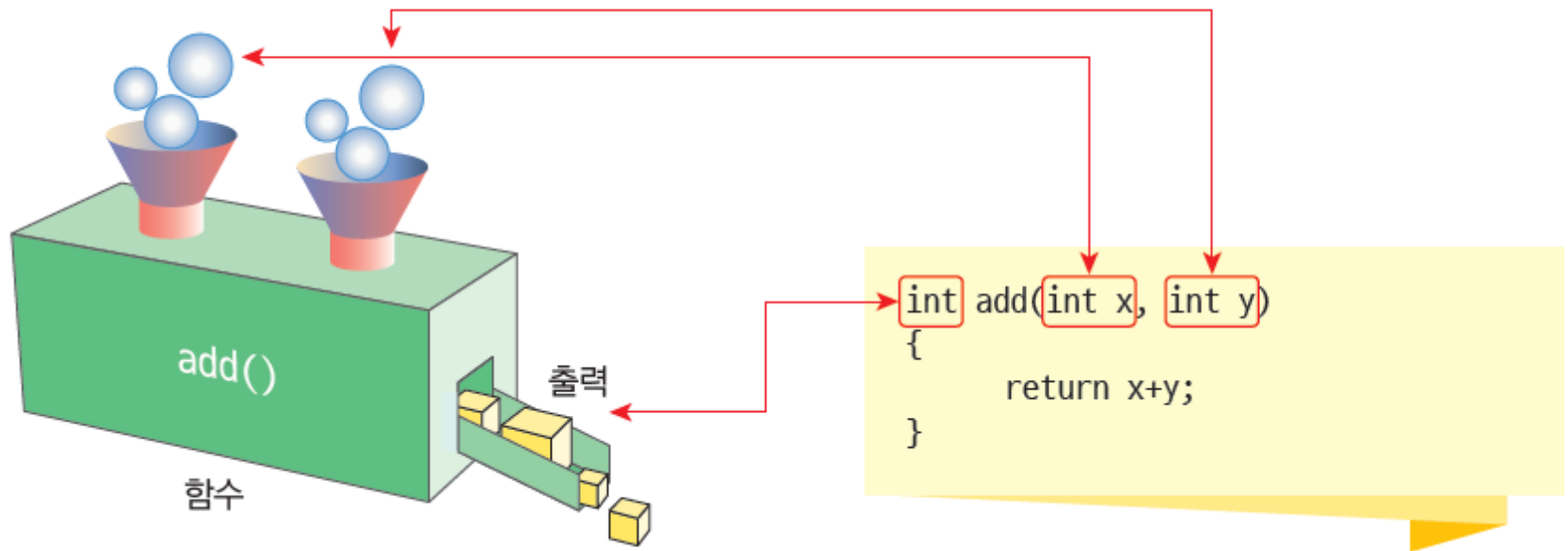
int get_integer(void)
{
    int n;

    printf("정수를 입력하시오: ");
    scanf("%d", &n);

    return n;
}
```



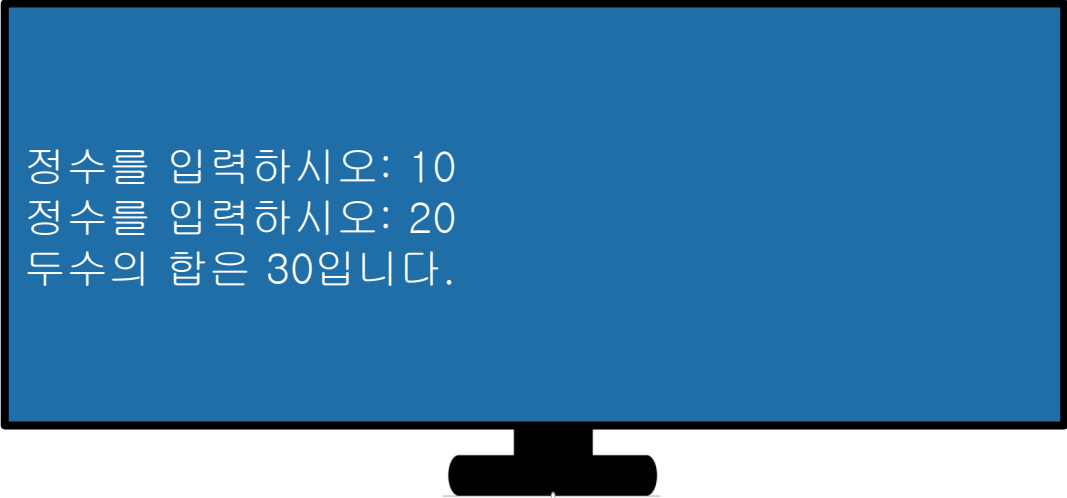
Lab: 정수의 합을 계산하는 add() 함수





Lab: 정수의 합을 계산하는 프로그램

- 앞에서 작성한 `get_integer()`까지 사용하여 사용자로 부터 받은 정수의 합을 계산하여 출력하자.



```
정수를 입력하시오: 10  
정수를 입력하시오: 20  
두수의 합은 30입니다.
```



```
#include <stdio.h>
//
int get_integer()
{
    int value;
    printf("정수를 입력하시오: ");
    scanf("%d", &value);
    return value;
}

//
int add(int x, int y)
{
    return x + y;
}

int main(void)
{
    int x = get_integer();
    int y = get_integer();

    int sum = add(x, y);
    printf("두수의 합은 %d입니다. \n", sum);
    return 0;
}
```




Lab: 팩토리얼 계산 함수

- 이번에는 정수 n 을 받아서 1에서 n 까지의 정수들의 곱을 구하는 팩토리얼 함수를 작성하여 보자.

$$n! = n * (n-1) * (n-2) * \dots * 1$$

알고 싶은 팩토리얼의 값은? 12
12!의 값은 479001600입니다.



예제

```
#include <stdio.h>

long factorial(int n)
{
    long result = 1;

    for (int i = 1; i <= n; i++)
        result *= i; // result = result * i
    return result;
}

int main(void)
{
    int n;
    printf("알고 싶은 팩토리얼의 값은? ");
    scanf("%d", &n);
    printf("%d!의 값은 %d입니다. \n", n, factorial(n));
    return 0;
}
```



Lab: 온도 변환기

'c' 섭씨온도에서 화씨온도로 변환
'f' 화씨온도에서 섭씨온도로 변환
'q' 종료
메뉴에서 선택하세요. f
화씨온도: 100
섭씨온도: 37.77778
'c' 섭씨온도에서 화씨온도로 변환
'f' 화씨온도에서 섭씨온도로 변환
'q' 종료
메뉴에서 선택하세요._



예제

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void printOptions()
{
    printf(" 'c' 섭씨온도에서 화씨온도로 변환\n");
    printf(" 'f' 화씨온도에서 섭씨온도로 변환\n");
    printf(" 'q' 종료\n");
}

double C2F(double c_temp)
{
    return 9.0 / 5.0 * c_temp + 32;
}

double F2C(double f_temp)
{
    return (f_temp - 32.0) * 5.0 / 9.0;
}
```



```
int main(void)
{
    char choice;
    double temp;
    while (1) {
        printOptions();
        printf("메뉴에서 선택하세요.");
        choice = getchar();
        if (choice == 'q') break;
        else if (choice == 'c') {
            printf("섭씨 온도: ");
            scanf("%lf", &temp);
            printf("화씨 온도: %lf \n", C2F(temp));
        }
        else if (choice == 'f') {
            printf("화씨 온도: ");
            scanf("%lf", &temp);
            printf("섭씨 온도: %lf \n", F2C(temp));
        }
        getchar(); // 엔터키 문자를 삭제하기 위하여 필요!
    }
    return 0;
}
```



Lab: 은행 계좌 구현하기

- 돈만 생기면 저금하는 사람을 가정하자. 이 사람을 위한 함수 `save(int amount)`를 작성하여 보자. 이 함수는 저금할 금액을 나타내는 인수 `amount`만을 받으며 `save(100)`과 같이 호출된다. `save()`는 정적 변수를 사용하여 현재까지 저축된 총액을 기억하고 있으며 한번 호출될 때마다 총 저축액을 다음과 같이 화면에 출력한다.

```
=====
입금   출금   잔고
=====
10000           10000
50000           60000
          10000  50000
30000           80000
=====
```



알고리즘

1. **while(1)**
2. 사용자로부터 아이디를 입력받는다.
3. 사용자로부터 패스워드를 입력받는다.
4. 만약 로그인 시도 횟수가 일정 한도를 넘었으면 프로그램을 종료한다.
5. 아이디와 패스워드가 일치하면 로그인 성공 메시지를 출력한다.
6. 아이디와 패스워드가 일치하지 않으면 다시 시도한다.



```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

// amount가 양수이면 입금이고 음수이면 출금으로 생각한다.
void save(int amount)
{
    static long balance = 0;

    if (amount >= 0)
        printf("%d \t\t", amount);
    else
        printf("\t %d \t", -amount);

    balance += amount;
    printf("%d \n", balance);
}
```




```
int main(void) {  
    printf("=====\n");  
    printf("입금 \t출금\t 잔고\n");  
    printf("=====\n");  
    save(10000);  
    save(50000);  
    save(-10000);  
    save(30000);  
    printf("=====\n");  
    return 0;  
}
```



Lab: 한번만 초기화하기

- 정적 변수는 한번만 초기화하고 싶은 경우에도 사용된다

```
init(): 네트워크 장치를 초기화합니다.  
init(): 이미 초기화되었으므로 초기화하지 않습니다.  
init(): 이미 초기화되었으므로 초기화하지 않습니다.
```



```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
```

```
void init();
```

```
int main(void)
{
    init();
    init();
    init();
    return 0;
}
```

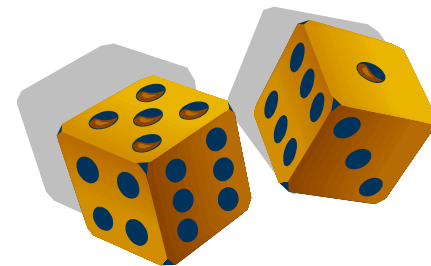
```
void init()
{
    static int initd = 0;
    if (initd == 0) {
        printf("init(): 네트워크 장치를 초기화합니다. \n");
        initd = 1;
    }
    else {
        printf("init(): 이미 초기화되었으므로 초기화하지 않습니다. \n");
    }
}
```



Lab: 난수 발생기

- 자체적인 난수 발생기 작성
- 이전에 만들어졌던 난수를 이용하여 새로운 난수를 생성함을 알 수 있다. 따라서 함수 호출이 종료되더라도 이전에 만들어졌던 난수를 어딘가에 저장하고 있어야 한다

$$r_{n+1} = (a \cdot r_n + b) \bmod M$$





실행 결과

- 0부터 10사이의 난수를 몇 개 생성해보자.

0 9 6 7 0 9 6 7 0 9



예제

random.c

```
#define SEED 17
int MULT = 25173;
int INC = 13849;
int MOD = 65536;

static unsigned int seed = SEED; // 난수 생성 시드값

// 정수 난수 생성 함수
unsigned random_i(void)
{
    seed = (MULT * seed + INC) % MOD; // 난수의 시드값 설정
    return seed;
}

// 실수 난수 생성 함수
double random_f(void)
{
    seed = (MULT * seed + INC) % MOD; // 난수의 시드값 설정
    return seed / (double)MOD; // 0.0에서 1.0 사이로 제한
}
```



예제

main.c

```
#include <stdio.h>

extern unsigned random_i(void);
extern double random_f(void);

extern int MOD;

int main(void)
{
    int i;

    MOD = 10;
    for (i = 0; i < 10; i++)
        printf("%d ", random_i());

    return 0;
}
```



Q & A

