

쉽게 풀어쓴 C언어 Express

제8장 함수



Prof. Jung Guk Kim
CESE, HUFS
jgkim@hufs.ac.kr

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.



모듈(Module)의 개념

■ 모듈(module)

- 독립되어 있는 프로그램의 일부분

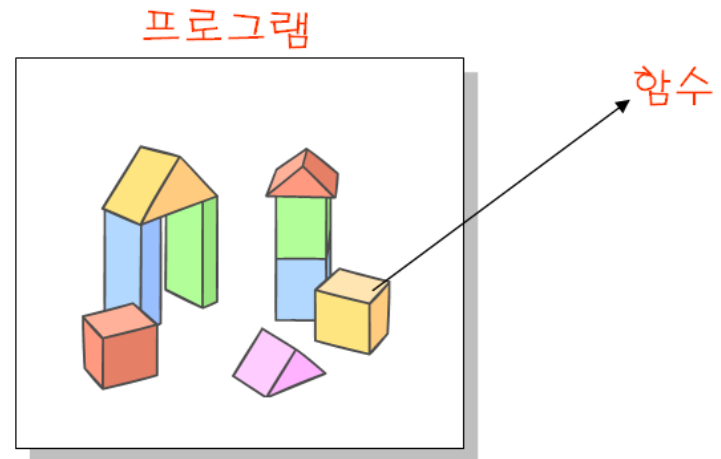
■ 모듈러 프로그래밍

- 여러 개의 모듈로 프로그래밍 구성: 개발자 역할 분담, communication 중요

■ 모듈러 프로그래밍의 장점

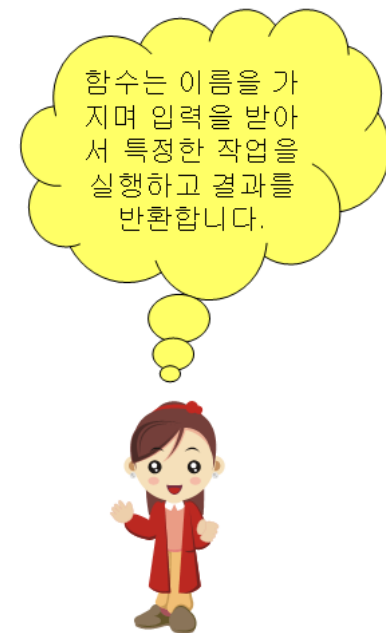
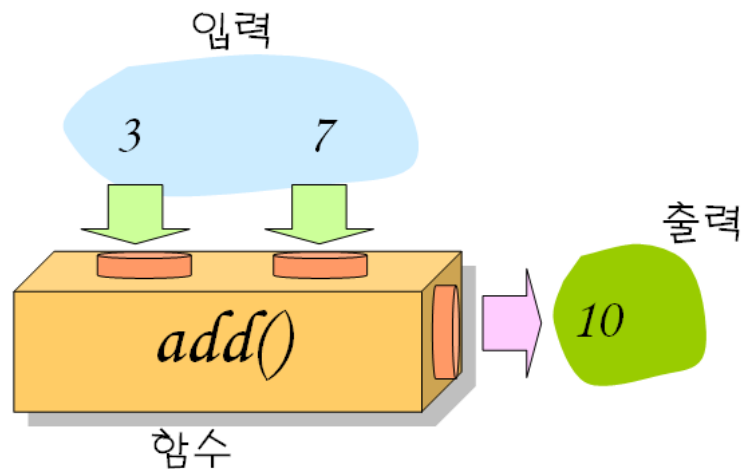
- 각 모듈들은 독자적으로 개발 가능
- 다른 모듈과 독립적으로 변경 가능
- 유지 보수가 쉬워진다: maintainable
- 모듈의 재사용 가능: reusability

■ C에서의 모듈 중 한 형태는 함수



함수(Function)의 개념

- 함수(function): 특정한 독립적 작업을 수행하는 프로그램 모듈
- 함수 호출(function call): 함수를 호출하여 사용하는 것
- 함수는 입력을 받으며 출력을 생성하여 호출자 (caller)에게 반환한다.



함수의 필요성 (반복)

```
#include <stdio.h>
int main(void)
{
    int i;
    for(i = 0; i < 10; i++)
        printf("*");
    ...
    for(i = 0; i < 10; i++)
        printf("*");
    ...
    for(i = 0; i < 10; i++)
        printf("*");
    return 0;
}
```

함수의 필요성 (반복)

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

```
void print_star()
```

```
{
```

```
    int i;
```

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

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

```
}
```

```
int main(void)
```

```
{
```

```
    print_star();
```

```
    ...
```

```
    print_star();
```

```
    ...
```

```
    print_star();
```

```
    return 0;
```

```
}
```

함수를 정의하였다. 함수는 한번 정의되면 여러 번 호출하여서 실행이 가능하다.

코드의 길이를 줄인다.

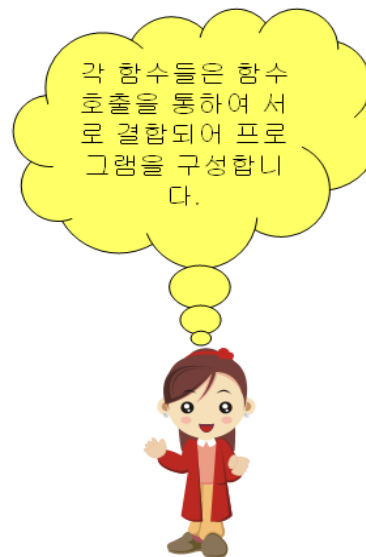
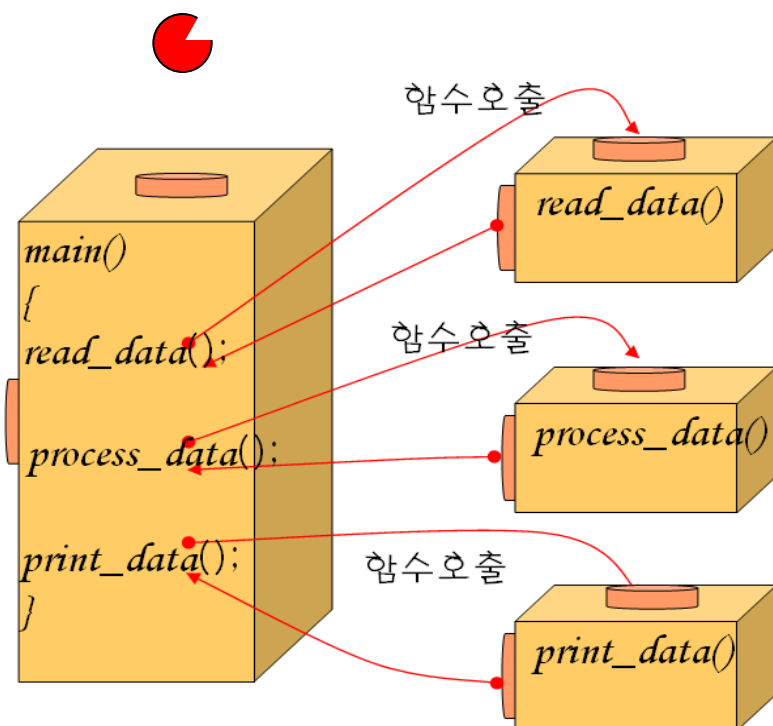
요약/추상화로 이해가 쉽다.

함수의 장점

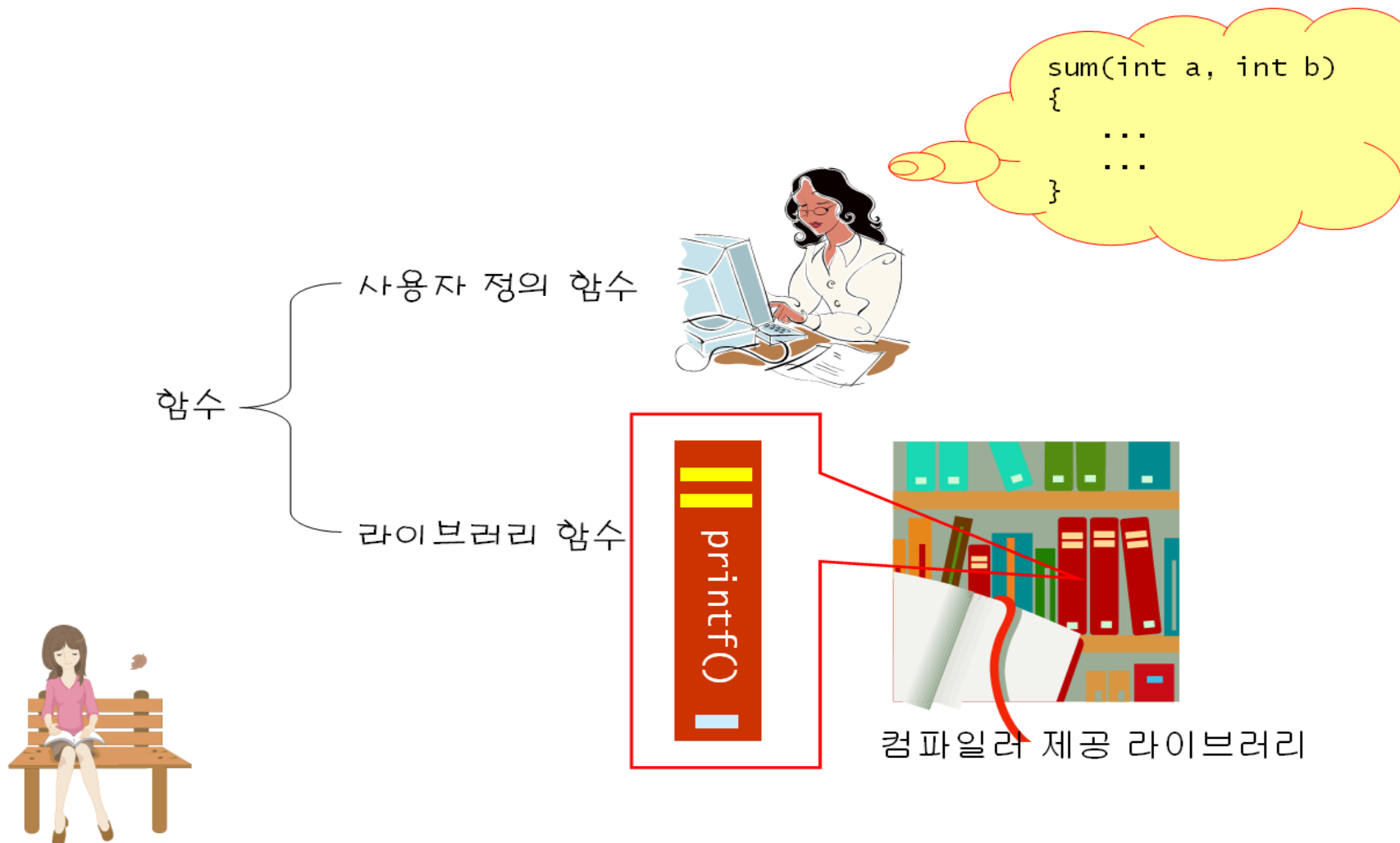
- 함수를 사용하면 코드가 중복되는 것을 막을 수 있다.
- 한번 작성된 함수는 여러 번 재사용할 수 있다.
- 함수를 사용하면 전체 프로그램을 모듈로 나눌 수 있어서 개발 과정이 쉬워지고 보다 체계적이 되면서 유지보수도 쉬워진다.
- **Input generality**
 - `sin(1.0); sin(2.0);`
- **Level of abstraction**
 - 논리 단계에 맞는 추상화

함수들의 연결

- 프로그램은 여러 개의 함수들로 이루어진다.
- 함수 호출을 통하여 서로 서로 연결된다.
- 운영체제에 의해 제일 먼저 호출되는 함수는 **main()**이다.



함수의 종류: 사용자 정의, 라이브러리



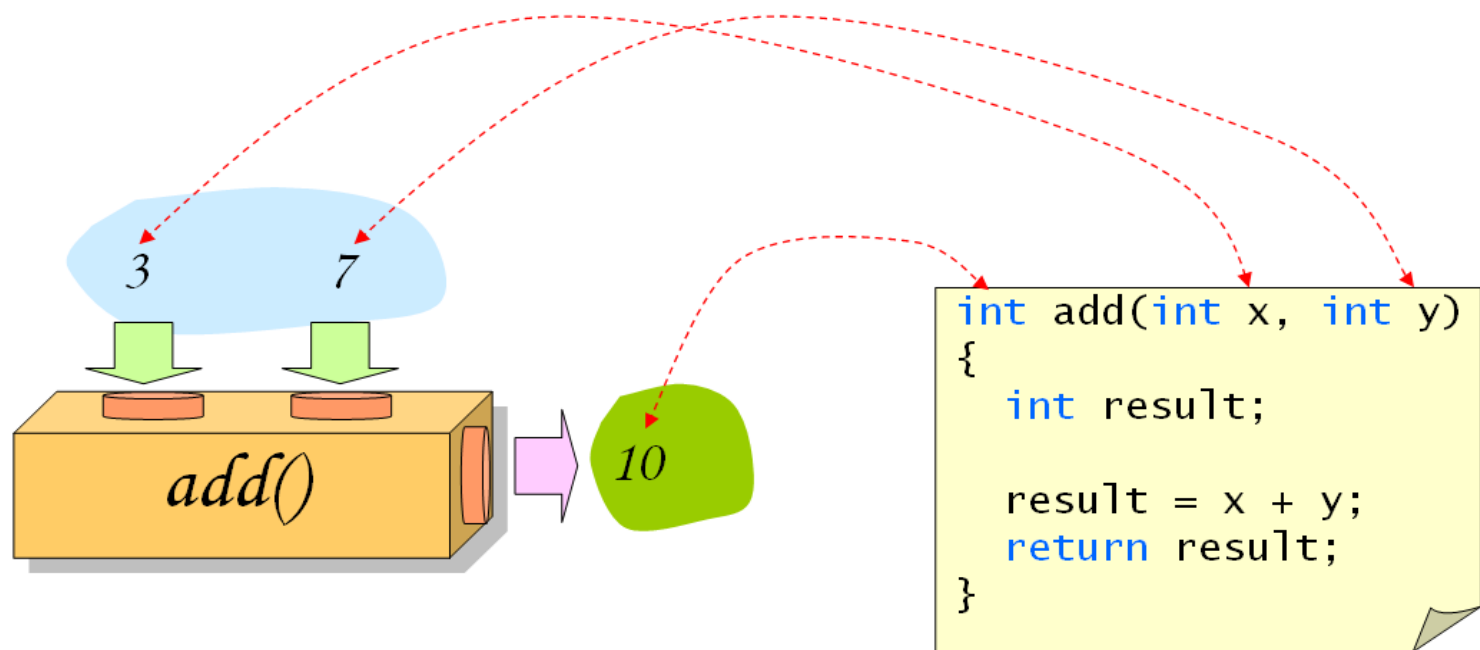
중간 점검

- 함수가 필요한 이유는 무엇인가?
- 함수와 프로그램의 관계는?
- 컴파일러에서 지원되는 함수를 _____ 함수라고 한다.
 - 위는 틀린 말.
 - 라이브러리는 누구나 만들어 제공이 가능하고, 운영체제에 포함되어 오는 것과 컴파일러에 포함된 것도 있다.
 - 컴파일러는 여러 함수들을 라이브러리에서 찾아서 사용자 프로그램과 합하여 주는데, 이를 linking 이라 한다.

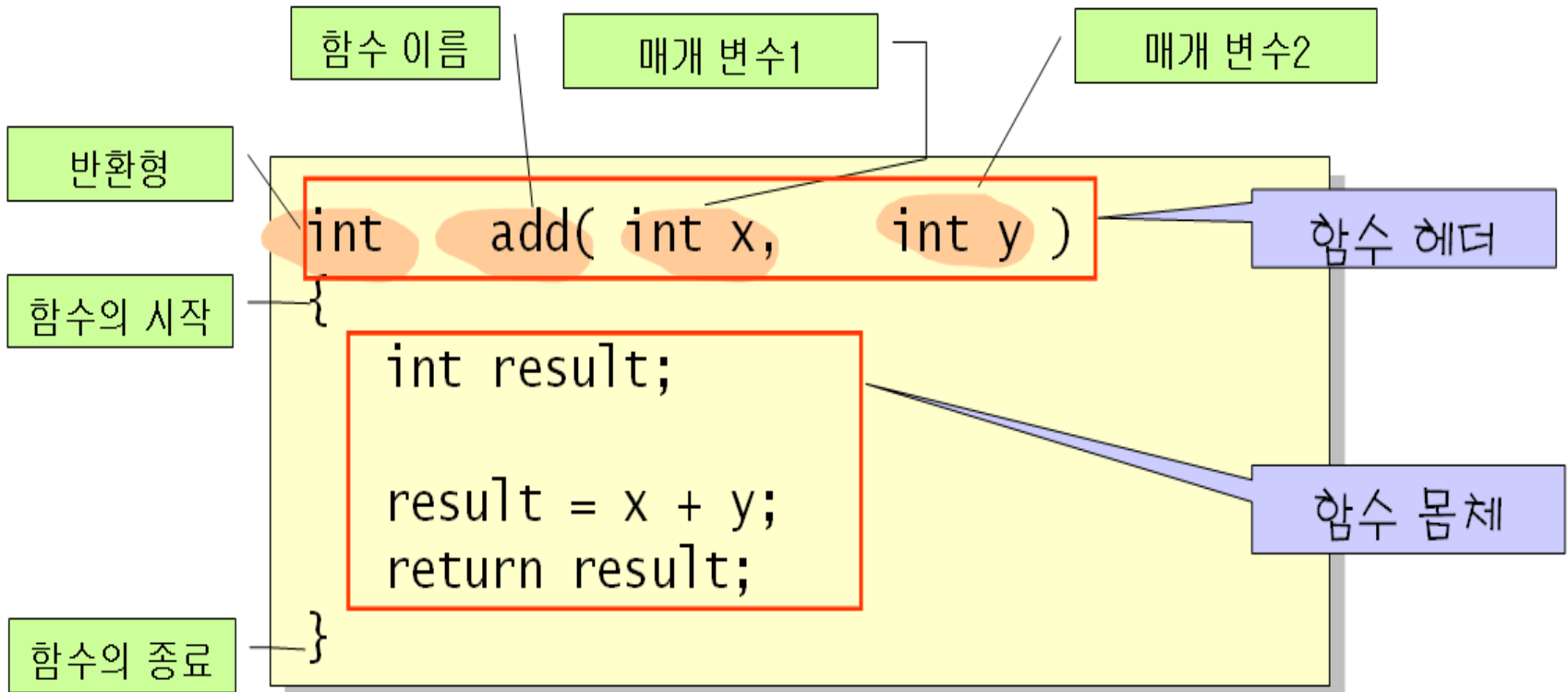


함수의 정의

1. 반환형(return type)
2. 함수 헤더(function header):
function name + parameters (arguments, 매개변수)
3. 함수 몸체(function body): 함수 프로그램



함수의 구조



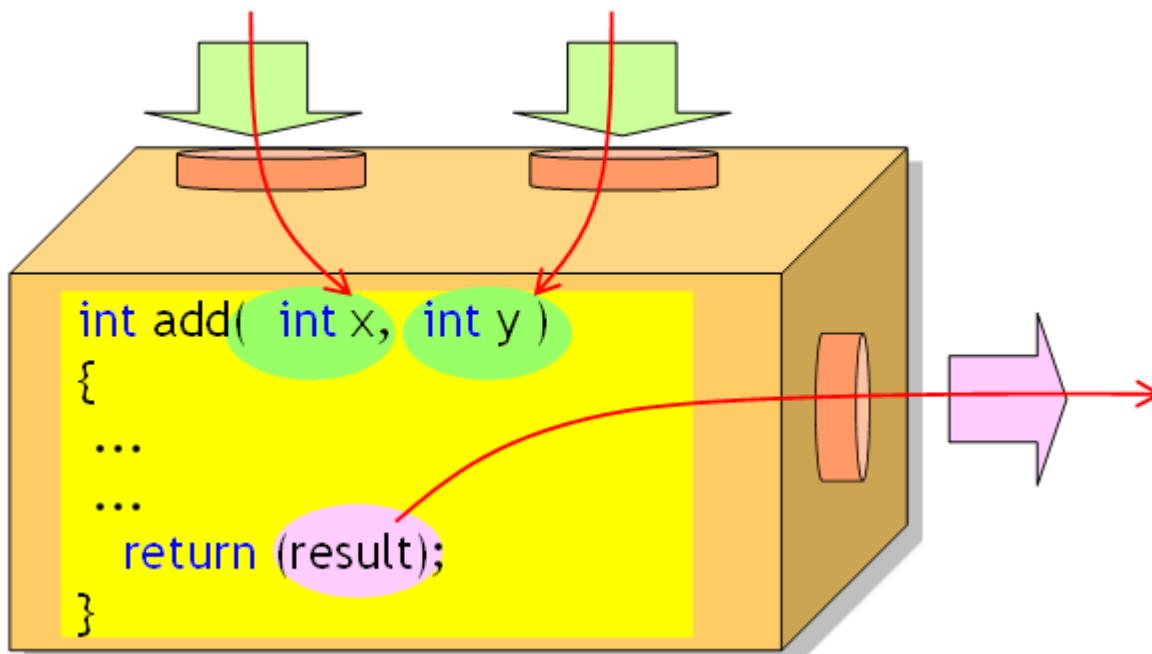
반환형

int
double
void

square()
compute_average()
set_cursor_type()

// int 형의 값을 반환한다.
// double 형의 값을 반환한다.
// 반환값이 없는 함수

반환형



매개 변수 (Arguments)

```
int square(int n)
```

```
double compute_average(double x, double y)
```

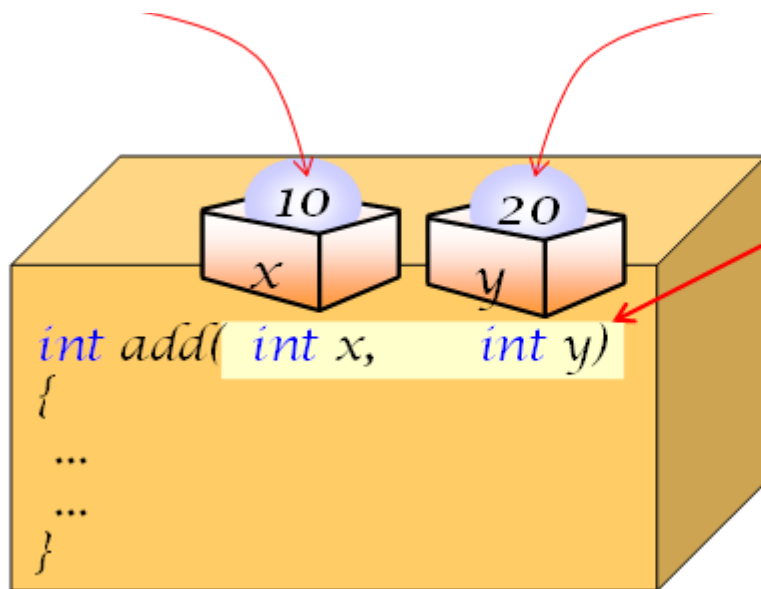
```
void get_cursor_type(void)
```

// 정수를 제공하는 함수

// 평균을 구하는 함수

// 커서의 타입을 반환하는 함수

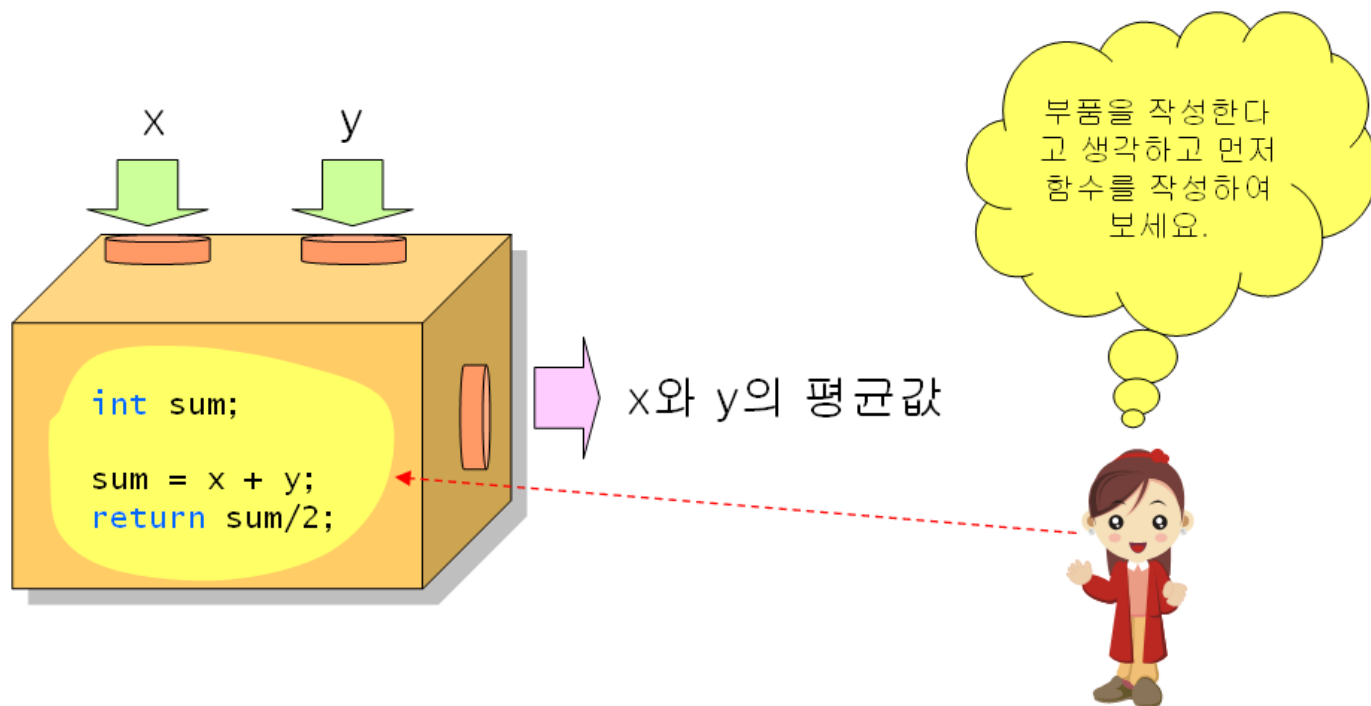
매개 변수



매개변수는 외부에서 전달되는 데이터가 저장되는 변수

함수 정의 예제

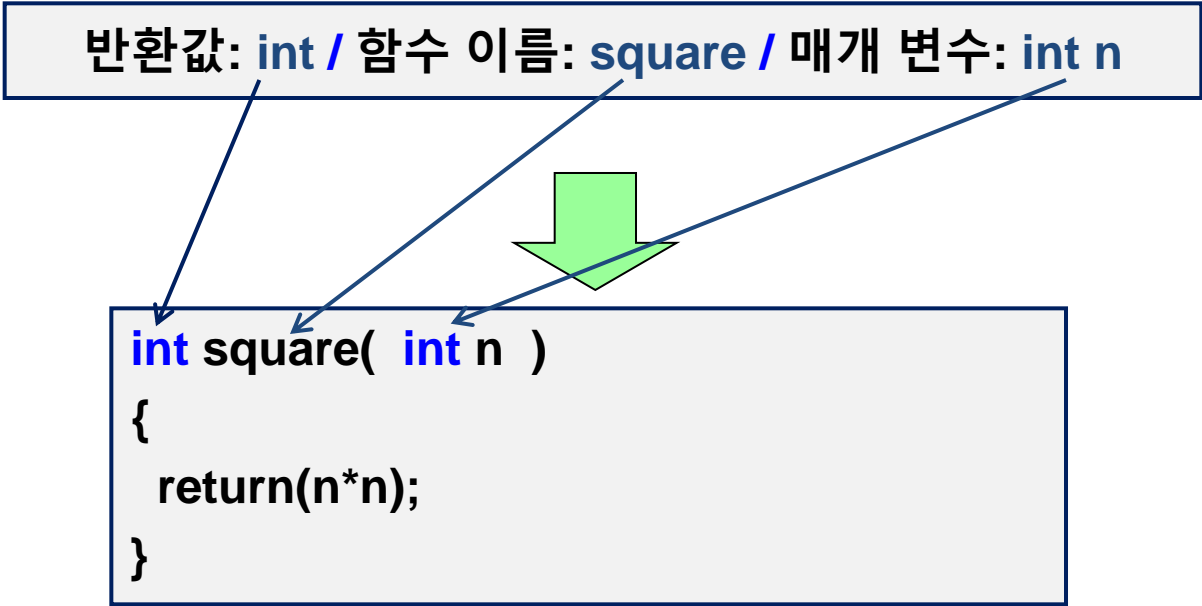
- 함수를 프로그램을 이루는 부품이라고 가정하자.
- 입력을 받아서 작업한 후에 결과를 생성한다.



예제 #1

- 정수의 제곱 값을 계산하는 함수

반환값: `int` / 함수 이름: `square` / 매개 변수: `int n`

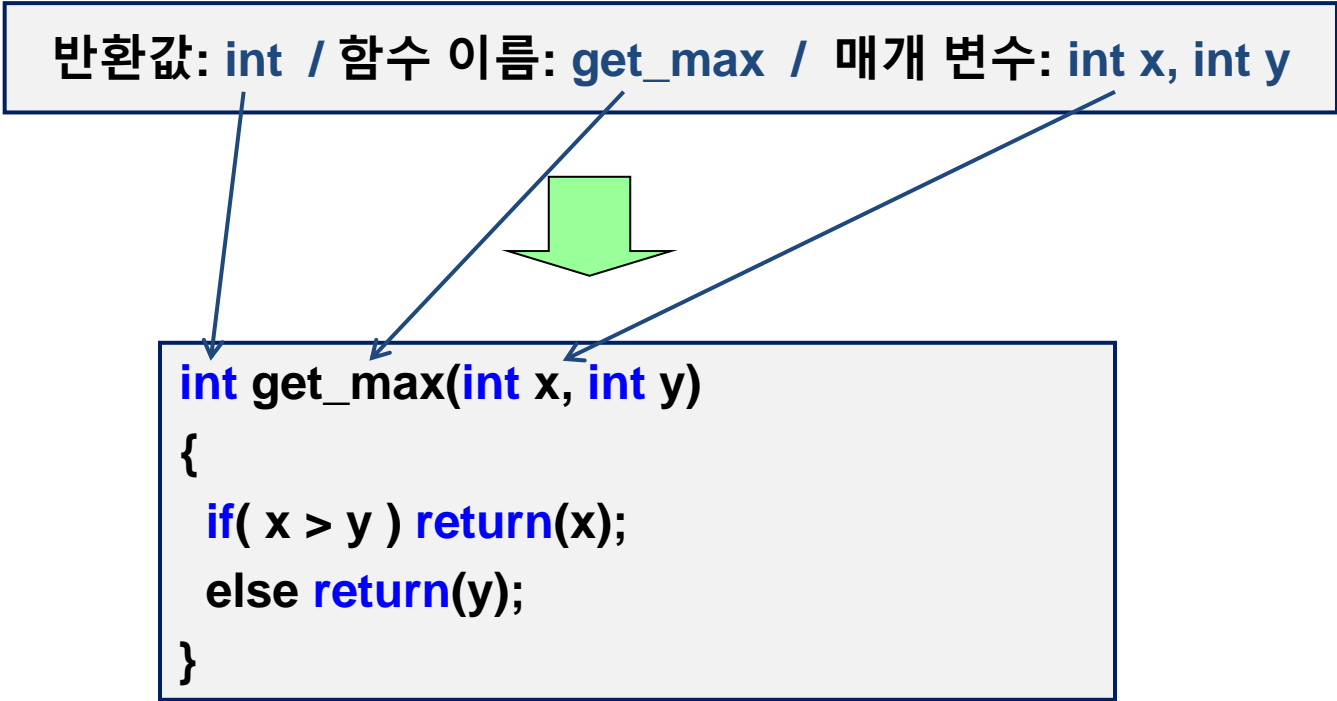


```
int square( int n )  
{  
    return(n*n);  
}
```


예제 #2

- 두개의 정수중에서 큰 수를 계산하는 함수

반환값: int / 함수 이름: get_max / 매개 변수: int x, int y

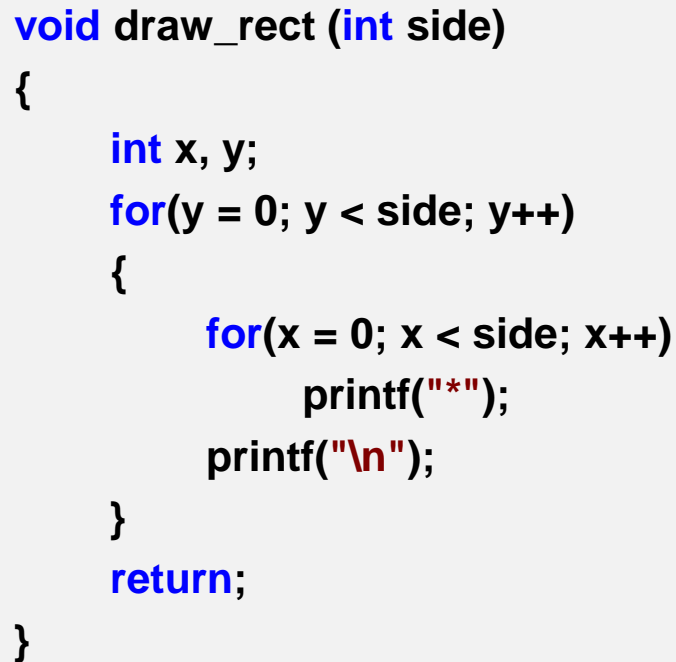


```
int get_max(int x, int y)
{
    if( x > y ) return(x);
    else return(y);
}
```

예제 #3

- 별표 기호를 이용하여 정사각형을 그리는 함수

반환값: **void** / 함수 이름: **draw_rect** / 매개 변수: **int side**

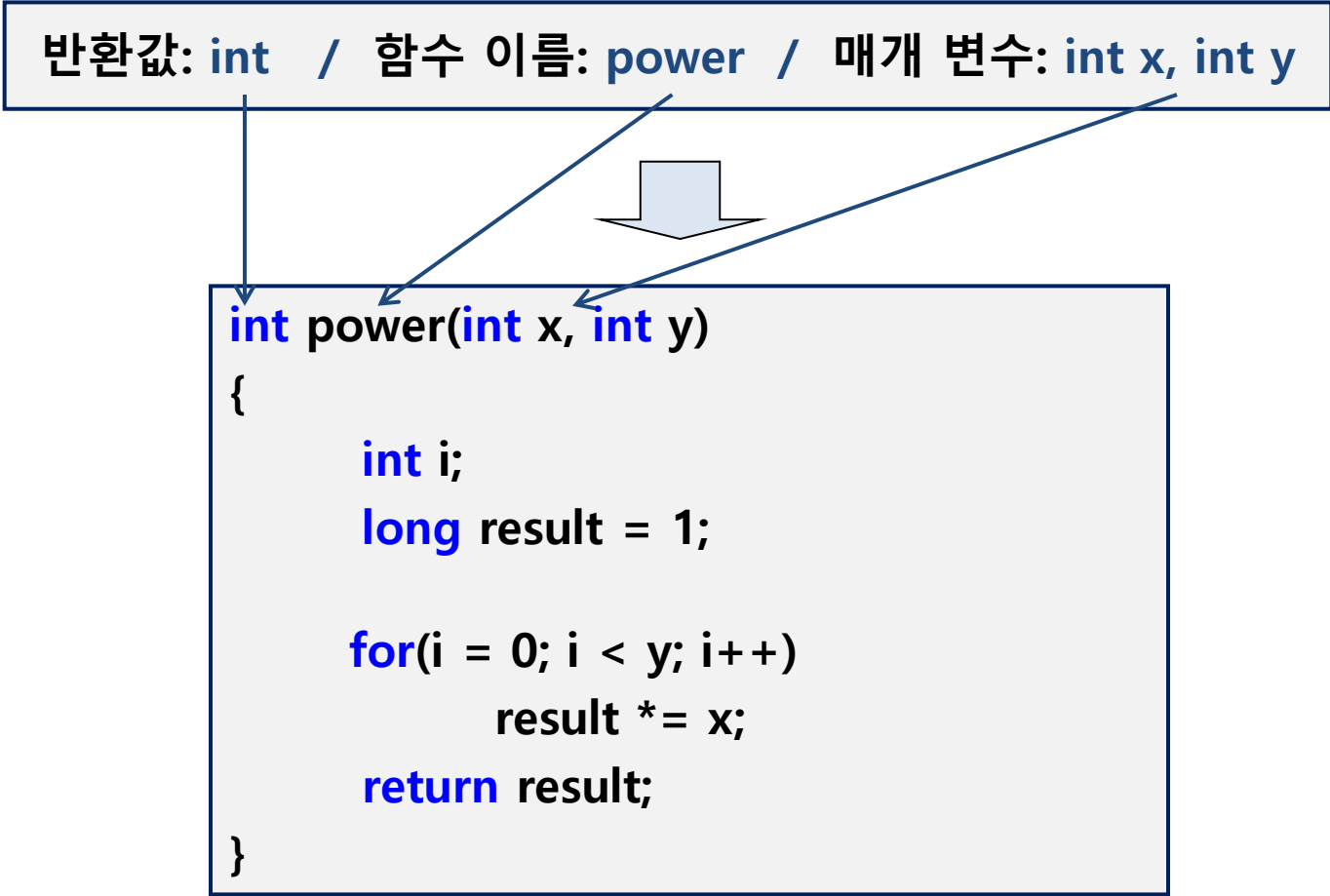


```
void draw_rect (int side)
{
    int x, y;
    for(y = 0; y < side; y++)
    {
        for(x = 0; x < side; x++)
            printf("*");
        printf("\n");
    }
    return;
}
```

예제 #4

- 정수의 거듭 제곱값(x^y)을 계산하는 함수

반환값: `int` / 함수 이름: `power` / 매개 변수: `int x, int y`



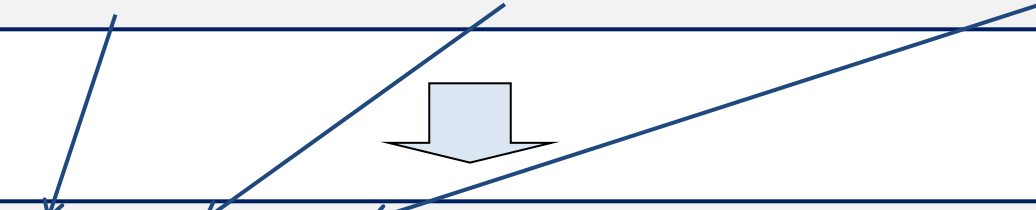
```
int power(int x, int y)
{
    int i;
    long result = 1;

    for(i = 0; i < y; i++)
        result *= x;
    return result;
}
```

예제 #5

- 팩토리얼값($n!$)을 계산하는 함수

반환값: `int` / 함수 이름: `factorial` / 매개 변수: `int n`



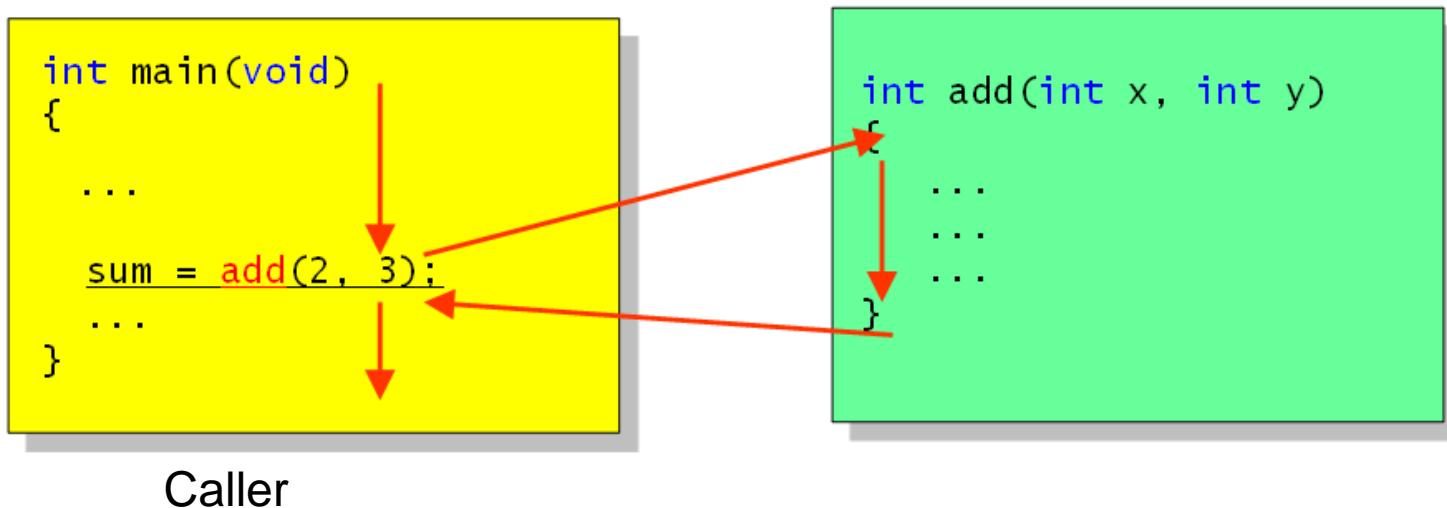
```
int factorial(int n)
{
    int i;
    long result = 1;

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

함수 호출과 반환

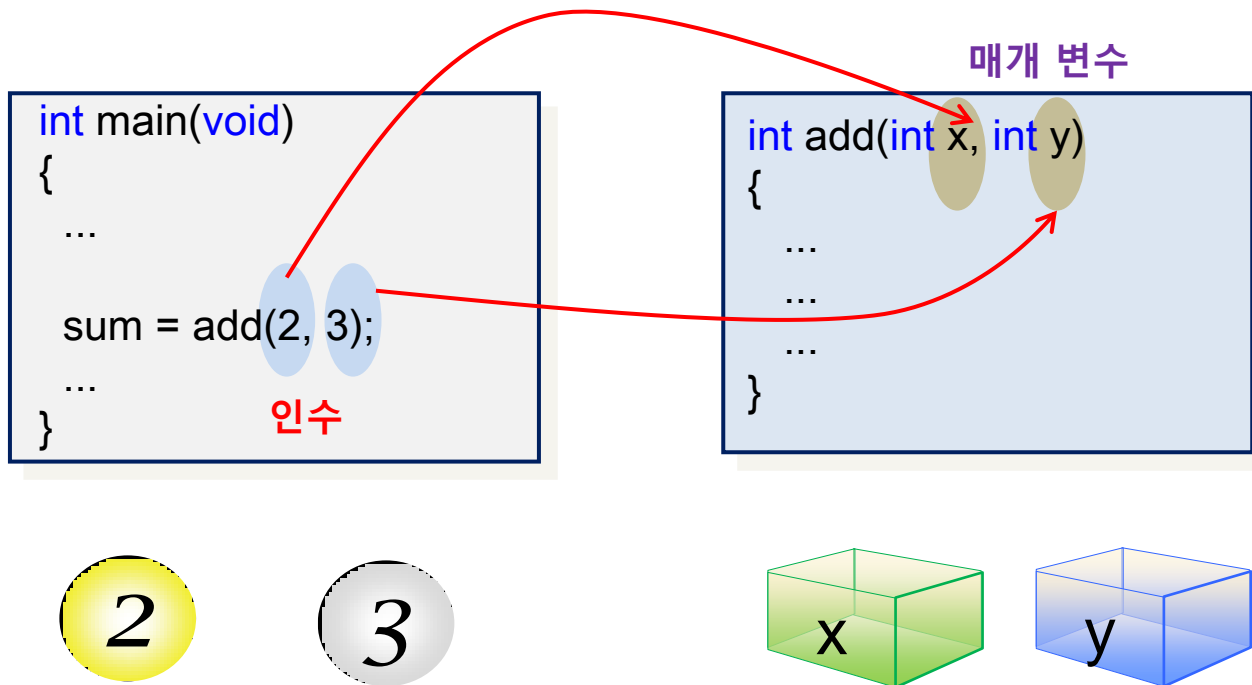
■ 함수 호출(function call):

- 함수를 사용하기 위하여 함수의 이름을 적어주는 것: call
- 다른 함수를 부르는 함수는 caller라 함
- 함수 안의 문장들이 순차적으로 실행된다.
- 함수의 실행이 끝나면 **호출한 위치로 되돌아 간다.**
- 결과값을 전달할 수 있다.



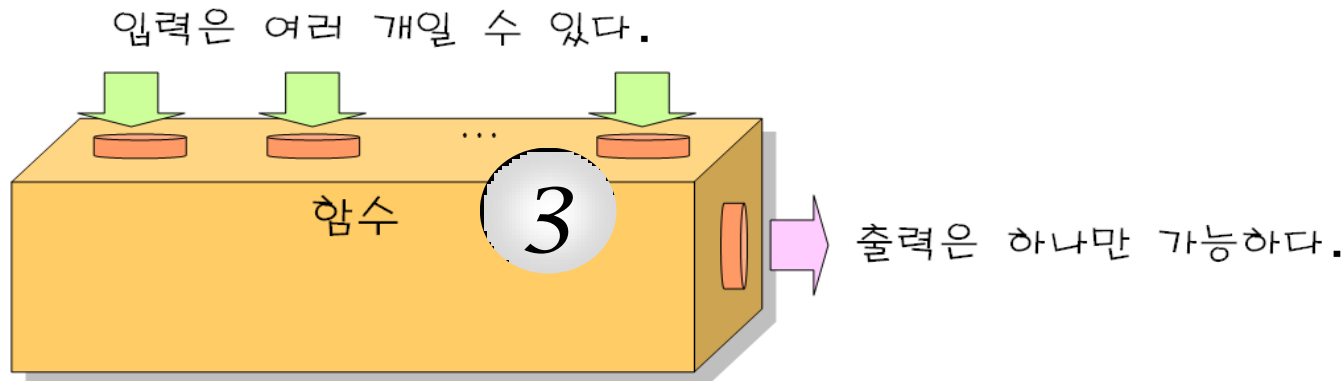
인수/매개 변수

- 인수(actual argument): 실인수, 실매개 변수라고도 한다.
- 매개 변수(formal parameter): 형식 인수, 형식 매개 변수라고도 한다.



반환값

- 반환 값(return value): 호출된 함수가 호출한 곳으로 작업의 **결과 값을 전달**하는 것
- 인수는 여러 개가 가능하나 반환 값은 **하나만 가능**



```
return 0;  
return(0);  
return x;  
return x*x+2*x+1;
```

반환 값

// 정수의 제곱을 계산하는 함수 예제

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

```
int square(int n);
```

```
int main(void)
```

```
{
```

```
    int result;
```

```
    result = square(5);
```

```
    printf("%d ", result);
```

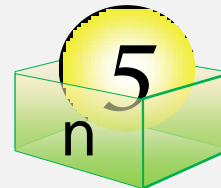
```
}
```

```
int square(int n)
```

```
{
```

```
    return(n * n);
```

```
}
```



반환 값

// 두수 중에서 큰 수를 찾는 함수 예제

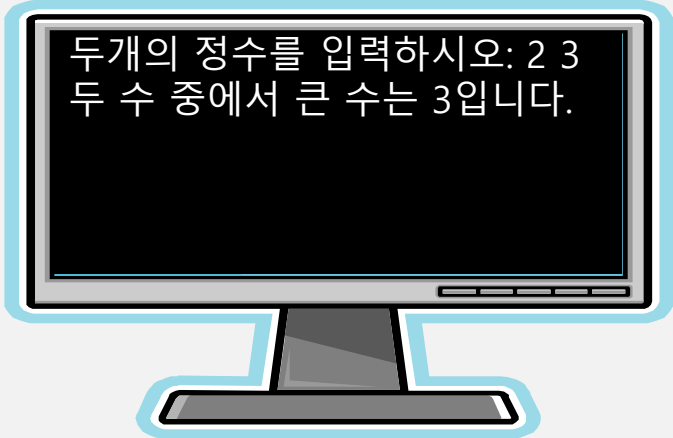
```
#include <stdio.h>
```

```
int get_max(int x, int y);
```

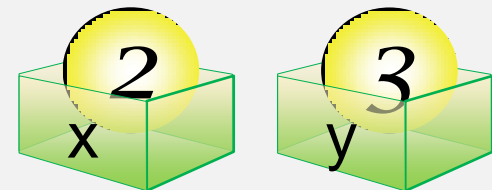
```
int main(void)
```

```
{  
    int a, b;  
    printf("두개의 정수를 입력하시오: ");  
    scanf("%d %d", &a, &b);  
    printf("두수 중에서 큰 수는 %d입니다.", get_max(a, b));  
    return 0;  
}
```

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



두개의 정수를 입력하시오: 2 3
두 수 중에서 큰 수는 3입니다.



예제 3

// 거듭 제곱 값을 구하는 예제

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

```
int get_integer(void);
```

```
int power(int x, int y);
```

```
int main(void)
```

```
{
```

```
    int a, b;
```

```
    a = get_integer();
```

```
    b = get_integer();
```

```
    printf("%d의 %d승은 %d입니다. ", a, b, power(a, b));
```

```
    return 0;
```

```
}
```

// 사용자로부터 값을 입력받아서 반환

```
int get_integer(void)
```

```
{
```

```
    int n;
```

```
    printf("정수를 입력하시오: ");
```

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

```
    return n;
```

```
}
```

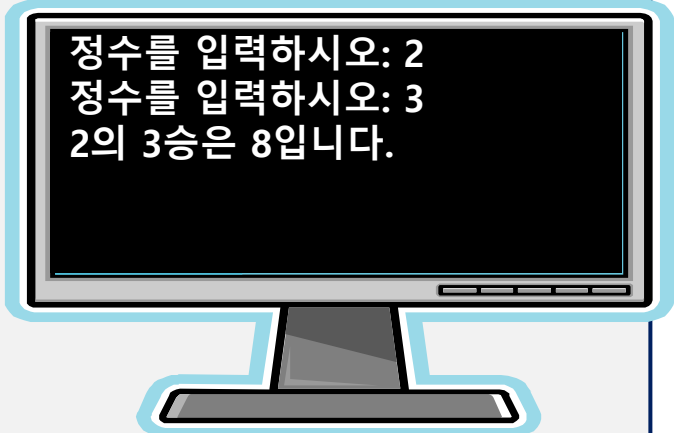
예제 3

// 거듭 제곱 값을 계산하여서 반환

```
int power(int x, int y)
{
    int i;
    long result = 1;    // 1로 초기화

    for(i = 0; i < y; i++)
        result *= x;    // result = result * x

    return result;
}
```



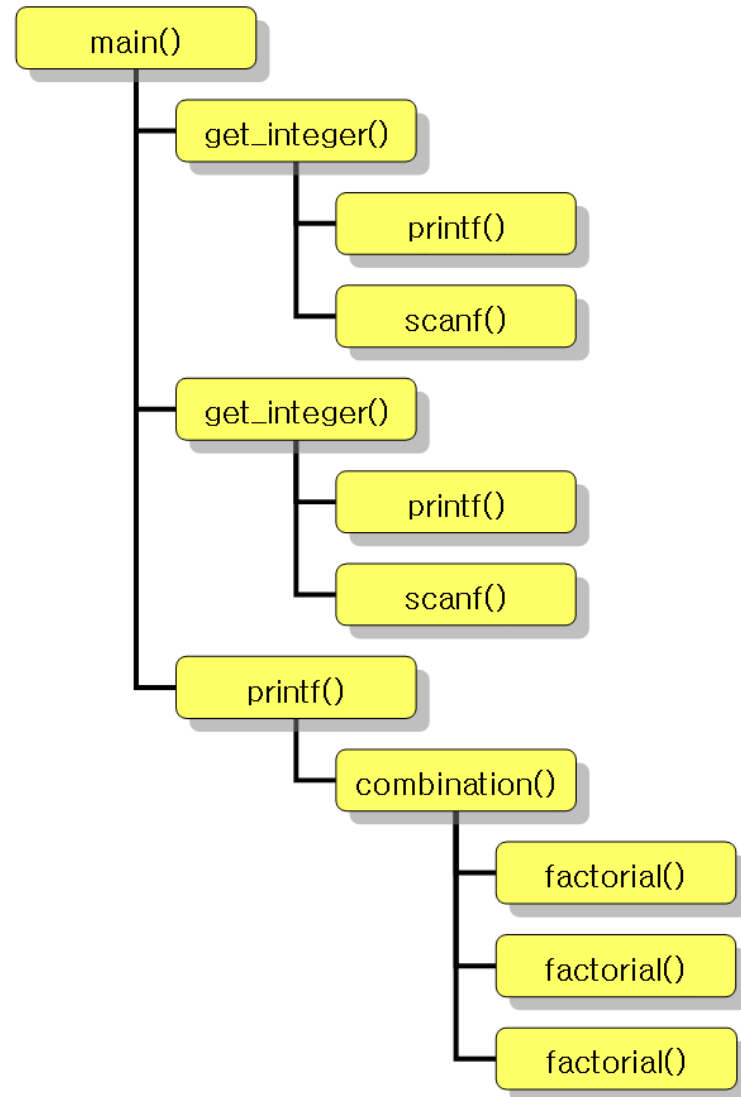
정수를 입력하시오: 2
정수를 입력하시오: 3
2의 3승은 8입니다.

조합(combination) 계산 함수

$$C(n, r) = \frac{n!}{(n-r)!r!}$$

$$C(3, 2) = \frac{3!}{(3-2)!2!} = \frac{6}{2} = 3$$

- 팩토리얼 계산 함수와 `get_integer()` 함수를 호출하여 조합을 계산한다



예제

```
#include <stdio.h>
// function prototypes to compile the main()
int get_integer(void);
int combination(int n, int r);
int factorial(int n);

int main(void)
{
    int a, b;

    a = get_integer();
    b = get_integer();

    printf("C(%d, %d) = %d \n", a, b, combination(a, b));
    return 0;
}

int combination(int n, int r)
{
    return (factorial(n)/(factorial(r) * factorial(n-r)));
}
```

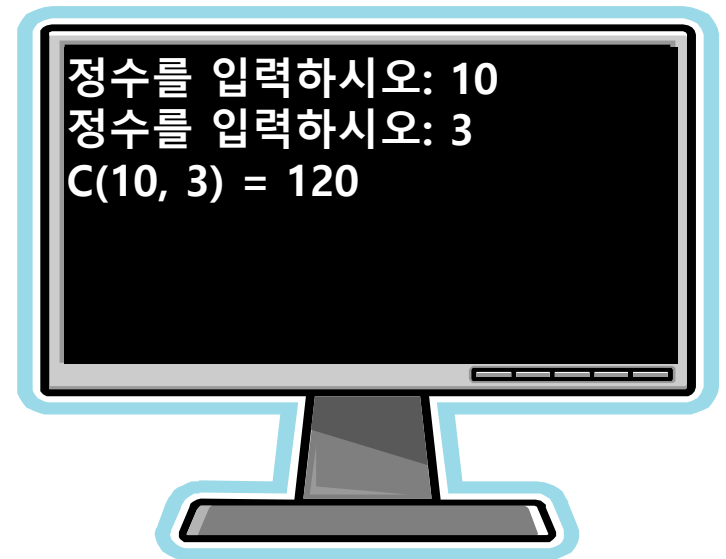
예제

```
int get_integer(void)
{
    int n;

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

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

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



Modular Programing

1. Top-down design / Top-down programming

- PM (Project Manager)는 하부 함수 모듈들을 정의하고,
- 함수가 없는 상태에서 main()을 작성한다.
- 함수는 각 하위 개발자에 일임하다.
- 추후에 통합한다. (integration)

2. Bottom-up design/programming

- 하위 함수 부터 작성하여 test 하고
- 작업 완료 후, main 을 작성한다.
- 하위 함수의 성공 여부가 불확실할 때 사용한다.

- 일반적으로 훌륭한 S/W architect 가 있다면 top-down structured modular program이 선호된다. (경비 절감, 유지보수)

- 큰 논리를 먼저 생각했기 때문에 전체적 설계의 오류가 적다.

중간 점검

- 인수와 매개 변수는 어떤 관계가 있는가?
- 반환 값이 실수로 정의된 함수에서 실수로 정수를 반환하면 어떤 일이 발생하는가?

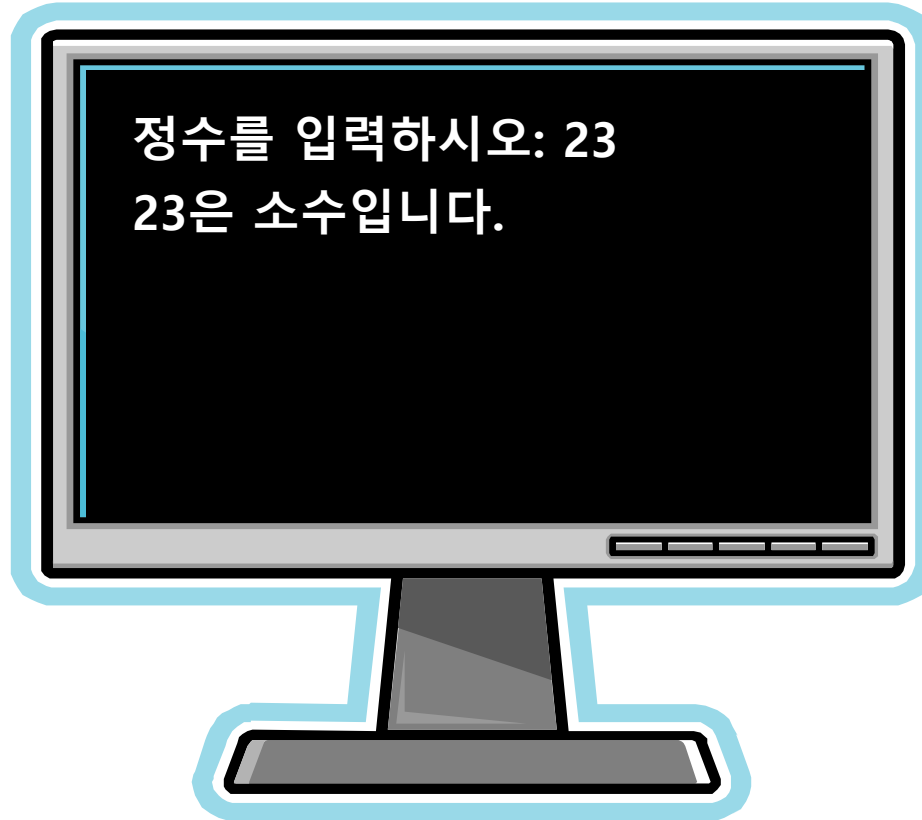


실습: 소수 찾기

- 주어진 숫자가 소수(prime number)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

실행결과



알고리즘

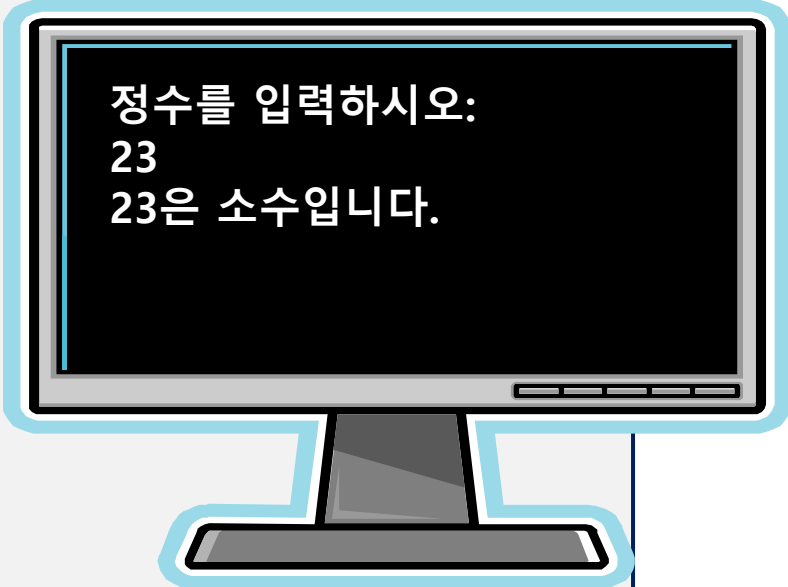
1. 사용자로부터 정수를 입력 받아서 변수 n 에 저장한다.
2. 약수의 개수를 0으로 초기화한다.
3. `for(i=1; i<=n ; i++)`
 1. n 을 i 로 나누어서 나머지가 0인지 본다.
 2. 나머지가 0이면 약수의 개수를 증가한다.
4. 약수의 개수가 2이면 정수 n 은 소수이다.



```
#include <stdio.h>
int is_prime(int);
int get_integer(void);
main()
{
    int n, result;
    n = get_integer();
    result = is_prime(n);
    if ( result == 1 )
        printf("%d은 소수입니다.\n", n);
    else
        printf("%d은 소수가 아닙니다.\n", n);
    return 0;
}
```

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

int is_prime(int n)
{
    int divisors = 0, i;
    for ( i = 1 ; i <= n ; i++ )
    {
        if ( n%i == 0 )
            divisors++;
    }
    return (divisors == 2);
}
```



정수를 입력하시오:
23
23은 소수입니다.

도전문제

- `is_prime()` 함수의 실행 속도를 빠르게 하기 위하여 어떤 코드를 추가할 수 있는지 생각해보자. 현재 버전은 검사하는 숫자가 매우 크면 비효율적이다. 예를 들어서 1,000,000에 대하여 호출되면 백만 번 반복을 하여야 한다. 한 가지 방법은 1보다 크고 n 보다 작은 숫자 중에서 약수가 하나라도 발견되면 이미 n 은 소수가 아니라고 생각하는 것이다. 이것을 코드로 작성하여 추가하여 보자.



함수 원형

- 함수 원형(function prototyping): 컴파일러에게 함수형에 대하여 미리 알리는 것

```
int compute_sum(int n);
```

```
int main(void)
```

```
{
```

```
    int sum;
```

```
    sum = compute_sum(100);
```

```
    printf("sum=%d \n", sum);
```

```
}
```

```
int compute_sum(int n)
```

```
{
```

```
    int i;
```

```
    int result = 0;
```

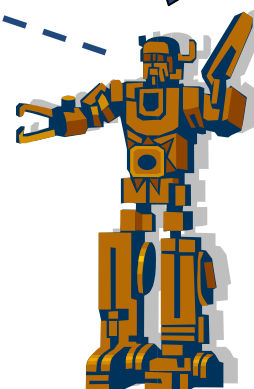
```
    for(i = 1; i <= n; i++)
```

```
        result += i;
```

```
    return result;
```

```
}
```

compute_sum()
은 함수
이름이랬지...



컴파일러

함수 원형의 형식

- 함수 원형(function prototype) : 미리 컴파일러에게 함수에 대한 정보를 알리는 것

반환형 함수이름(매개변수1, 매개변수2, ...);

- (예)
- int get_integer(void);
- int combination(int n, int r);

- (예)
- int get_integer(void);
- int combination(int, int);

자료형만 적어주어도 됨!



함수 원형을 사용하지 않는 예제

```
int compute_sum(int n)
{
    int i;
    int result = 0;
    for(i = 1; i <= n; i++)
        result += i;
    return result;
}
```

```
int main(void)
{
    int sum;
    sum = compute_sum(100);
    printf("sum=%d \n", sum);
}
```

함수 정의가 함수 호출보다 먼저 오면 함수 원형을 정의하지 않아도 된다.

그러나 일반적인 방법은 아니다.

함수 원형과 헤더 파일

- 보통은 헤더 파일에 함수 원형이 선언되어 있음

```
/* 두개의 숫자의 합을 계산하는 프로그램 */
#include <stdio.h>

int main(void)
{
    int n1;    /* 첫번째 숫자 */
    int n2;    /* 두번째 숫자 */
    int sum;   /* 두개의 숫자의 합을 저장 */

    printf("첫번째 숫자를 입력하시오:");
    scanf("%d", &n1);

    printf("두번째 숫자를 입력하시오:");
    scanf("%d", &n2);

    sum = n1 + n2;
    printf("두수의 합: %d", sum);

    return 0;
}
```

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

...
_CRTIMP int __cdecl printf(const char
*, ...);
...
_CRTIMP int __cdecl scanf(const char
*, ...);
...
```

stdio.h

add.c

중간 점검

- 함수 정의의 첫 번째 줄에는 어떤 정보들이 포함되는가? 이것을 무엇이라고 부르는가?
- 함수가 반환할 수 있는 값의 개수는?
- 함수가 값을 반환하지 않는다면 반환형은 어떻게 정의되어야 하는가?
- 함수 정의와 함수 원형의 차이점은 무엇인가?
- 함수 원형에 반드시 필요한 것은 아니지만 대개 매개 변수들의 이름을 추가하는 이유는 무엇인가?
- 다음과 같은 함수 원형을 보고 우리가 알 수 있는 정보는 어떤 것들인가?
- `double pow(double, double);`



라이브러리 함수

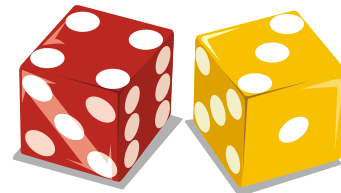
■ 라이브러리 함수(library function)

- 표준 입출력 (standard I/O library) : printf, scanf
- 수학 연산 (mathematical library) : sin, cos
- 문자열 처리 (string library) : strlen
- 시간 처리
- 오류 처리
- 데이터 검색과 정렬



난수 함수 (Random Number)

- 난수(random number)는 규칙성이 없이 임의로 생성되는 수이다.
- 난수는 암호학이나 시뮬레이션, 게임 등에서 필수적이다.
- rand()
 - 난수를 생성하는 함수
 - 0부터 RAND_MAX(32767)까지의 난수를 생성



예제: 로또 번호 생성하기

- 1부터 45번 사이의 난수 발생



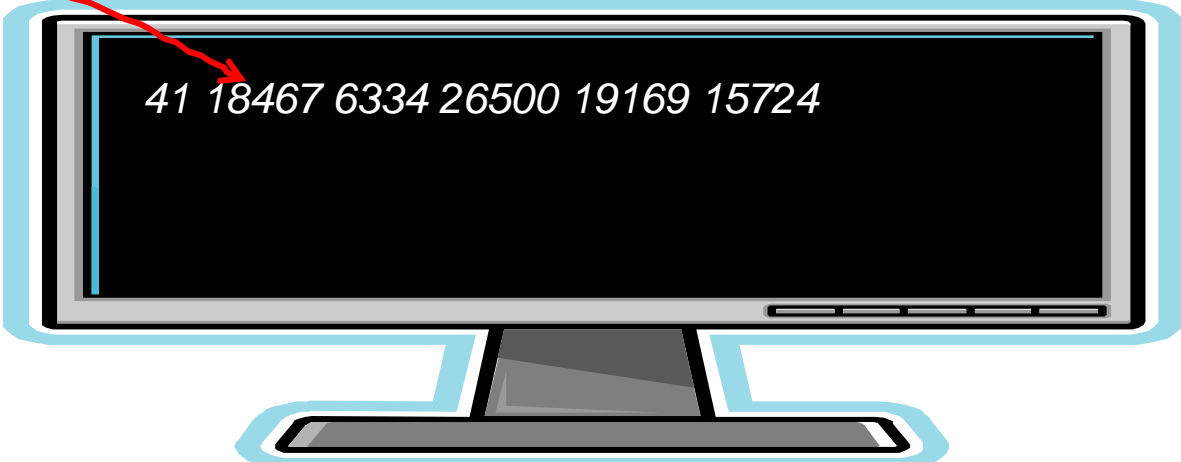
4 21 22 34 37 38 + 보너스번호 33 내 번호 당첨조회

실습 코드

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i;
    for(i = 0; i < 6; i++)
        printf("%d ", rand());

    return 0;
}
```

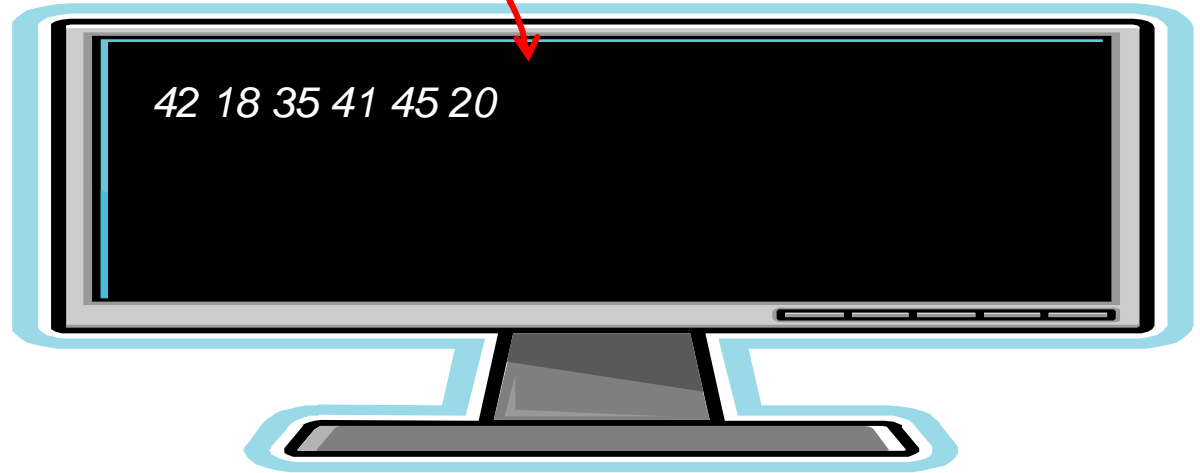
0에서 32767 사이의 정수로
생성



41 18467 6334 26500 19169 15724

1부터 45 사이로 제한된 난수의 생성

- `printf("%d ", 1+(rand()%45));`



- 하지만 매 프로그램 실행 시 마다 항상 똑같은 난수가 발생된다. (예: 게임: 똑 같은 게임 instance -> X)

실행할 때마다 다르게 하려면

- 매번 난수를 다르게 생성하려면 시드(seed)를 다르게 하여야 한다.

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

#define MAX 45
int main( void )
{
    int i;

    srand( (unsigned) time( NULL ) );
    for( i = 0; i < 6; i++ )
        printf("%d ", 1+rand()%MAX );
    return 0;
}
```

시드를 설정하는 가장 일반적인 방법은 현재의 시각을 시드로 사용하는 것이다. 현재 시각은 실행할 때마다 달라지기 때문이다.

실습: 자동차 게임

- 난수를 이용하여서 자동차 게임을 작성
- 사용자가 키를 누를 때마다 1초씩 주행하도록 하자.
- 주행 거리는 난수로 결정된다.



실행 결과

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

알고리즘

1. 난수 발생기를 초기화한다. (sedd 값 설정)
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 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 발생한다. 난수의 범위는 %연산자를 사용하여 0에서 99로 제한하였다.



```
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");
}
```

도전문제

- 위의 프로그램을 참고하여서 숫자야구 게임을 작성해보자. 숫자 야구 게임은 1~9 까지의 숫자 중에서 3개를 뽑아서 문제를 낸다. 단 숫자가 중복되면 안 된다.
- 예를 들어 029라고 하자. 사용자는 이 숫자를 맞추게 된다. 각 자리수와 숫자가 모두 일치하면 스트라이크, 숫자만 맞으면 볼이라고 출력한다.

029 vs 092 -> 1스트라이크 2볼



시스템 유틸리티 함수

함수	설명
exit(int status)	exit()를 호출하면 호출 프로세스를 종료시킨다.
int system(const char *command)	system()은 문자열 인수를 운영체제의 명령어 셸에게 전달하여서 실행시키는 함수이다.

```
#include <stdlib.h>
#include <stdio.h>
int main( void )
{
    system("dir"); // Linux: system("ls")
    printf("아무 키나 치세요\n");
    getch();
    system("cls");
    return 0;
}
```

C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: 507A-3B27
c:\source\chapter02\hello\hello 디렉터리
2011-11-28 오후 04:32 <DIR> .
2011-11-28 오후 04:32 <DIR> ..
2011-11-16 오전 11:01 20 binary.bin
...
4개 파일 5,296 바이트
3개 디렉터리 69,220,450,304 바이트 남음
아무 키나 치세요

수학 라이브러리 함수 (Math. Library)

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}

예제

// 삼각 함수 라이브러리

#include <math.h>

#include <stdio.h>

여러 수학 함수들을 포함하는 표준
라이브러리

int main(void)

{

double pi = 3.1415926535;

double x, y;

x = pi / 2;

y = sin(x);

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

y = sinh(x);

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

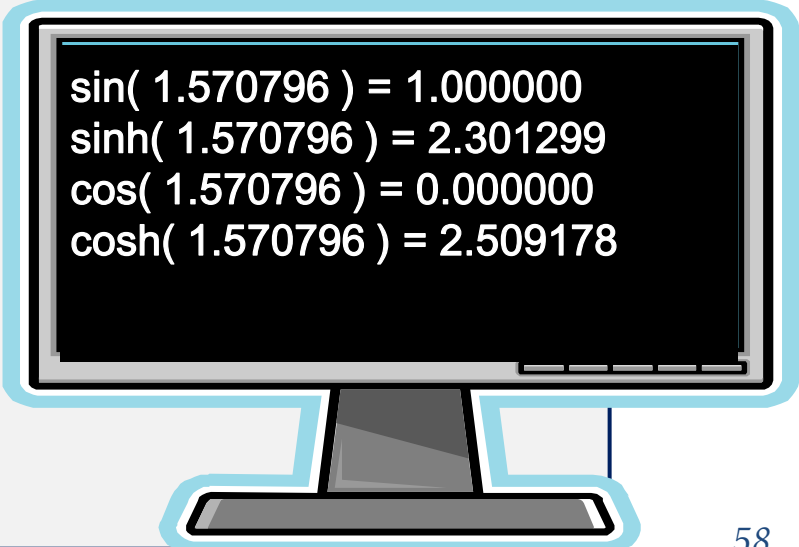
y = cos(x);

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

y = cosh(x);

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

}



```
sin( 1.570796 ) = 1.000000
sinh( 1.570796 ) = 2.301299
cos( 1.570796 ) = 0.000000
cosh( 1.570796 ) = 2.509178
```

예제

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

```
#define RAD_TO_DEG (45.0/atan(1))
```

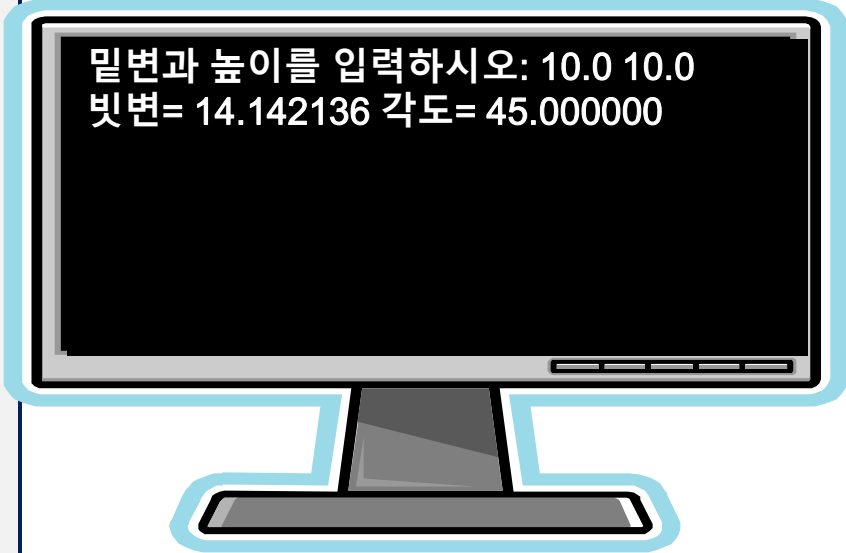
```
int main(void)
{
    double w, h, r, theta;

    printf("밑변과 높이를 입력하십시오:");
    scanf("%lf %lf", &w, &h);

    r = sqrt(w * w + h * h);
    theta = RAD_TO_DEG * atan2(h, w);

    printf("빗변= %f 각도= %f\n", r, theta);
    return 0;
}
```

상수를 정의하는 전처리
명령문



밑변과 높이를 입력하십시오: 10.0 10.0
빗변= 14.142136 각도= 45.000000

수학 라이브러리 함수들

- `abs(int x), fabs(double x)`
 - `abs(-9)` // 9를 반환
 - `fabs(-3.67)` // 3.67을 반환
- `pow(double x, double y)`
 - 인수 x 의 y - 거듭제곱인 x^y 을 계산한다.
 - `pow(2.0, 3.0);` // 8.0을 반환
- `sqrt(double x)`
 - 주어진 수의 제곱근을 구한다. 만약에 음수가 입력되면 오류가 발생한다.
 - `sqrt(9.0);` // 3.0을 반환
- `ceil(double x)`
 - `ceil`은 x 보다 작지 않은 가장 작은 정수를 반환
 - `ceil(-2.9);` // -2.0을 반환
 - `ceil(2.9);` // 3.0을 반환
- `floor(double x)`
 - `floor()`는 x 보다 크지 않은 가장 큰 정수를 반환한다.
 - `floor(-2.9);` // -3.0을 반환
 - `floor(2.9);` // 2.0을 반환

중간 점검

- 90도에서의 싸인 값을 계산하는 문장을 작성하여 보라.
- `rand() % 10` 이 계산하는 값의 범위는?



함수를 사용하는 이유

- 소스 코드의 중복을 없애준다.
 - 한번 만들어진 함수를 여러 번 호출하여 사용할 수 있다.
- 한번 작성된 함수를 다른 프로그램에서도 사용할 수 있다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.

```
void print_heading(void)
{
    printf("*****");
    printf(" NAME ADDRESS PHONE ");
    printf("*****");
}
```

```
int main(void)
{
    // 출력이 필요한 위치 #1
    print_heading();
    ...
    // 출력이 필요한 위치 #2
    print_heading();
    ...
    ...
}
```

```
int main(void)
{
    ...
    read_list();
    sort_list();
    print_list();
    ...
}
```