



AJOU UNIVERSITY



컴퓨터 프로그래밍 및 실습

6주차. 함수

실습 안내

■ 실습 제출 안내

- 솔루션 이름은 "Practice week 6"
- 프로젝트 이름과 소스코드 이름은 Problem1, Problem2, ...
 - 실습1의 프로젝트 이름은 Problem1, 소스코드 이름은 problem1.c
 - 실습 2의 프로젝트 이름은 Problem2, 소스코드 이름은 problem2.c ...
- 솔루션 폴더를 압축하여 **Practice_week6_학번_이름.zip** 으로 제출
- 제출기한: 당일 **19시** 까지
- 실습 관련 코드: <https://github.com/lani009/Ajou-c-programming>

1. 함수의 의미

❖ 함수(function)란 프로그램에서 특정한 작업을 수행하도록 따로 정해 놓은 독립된 단위

- 입력을 받아서 특정한 작업을 수행하여서 결과를 반환하는 문장들의 집합

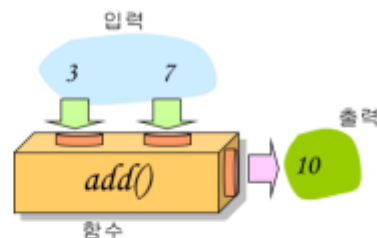
❖ 함수의 종류



1. 함수의 의미

❖ 함수의 특징

- 함수는 특정한 작업을 수행하기 위한 명령어들의 모음
- 함수는 서로 구별되는 이름을 가지고 있음
- 함수는 특정한 작업을 수행
- 함수는 입력을 받을 수 있고 결과를 반환할 수 있음



❖ 함수의 장점

- 함수를 사용하면 코드가 중복되는 것을 막을 수 있음
- 한번 작성된 함수는 여러 번 재사용할 수 있음
- 복잡한 문제를 단순한 부분으로 분해할 수 있음
 - ✓ 함수를 사용하면 전체 프로그램을 모듈로 나눌 수 있어서 개발 과정이 쉬워지고 보다 체계적이 되면서 유지보수도 쉬움

1. 함수의 의미

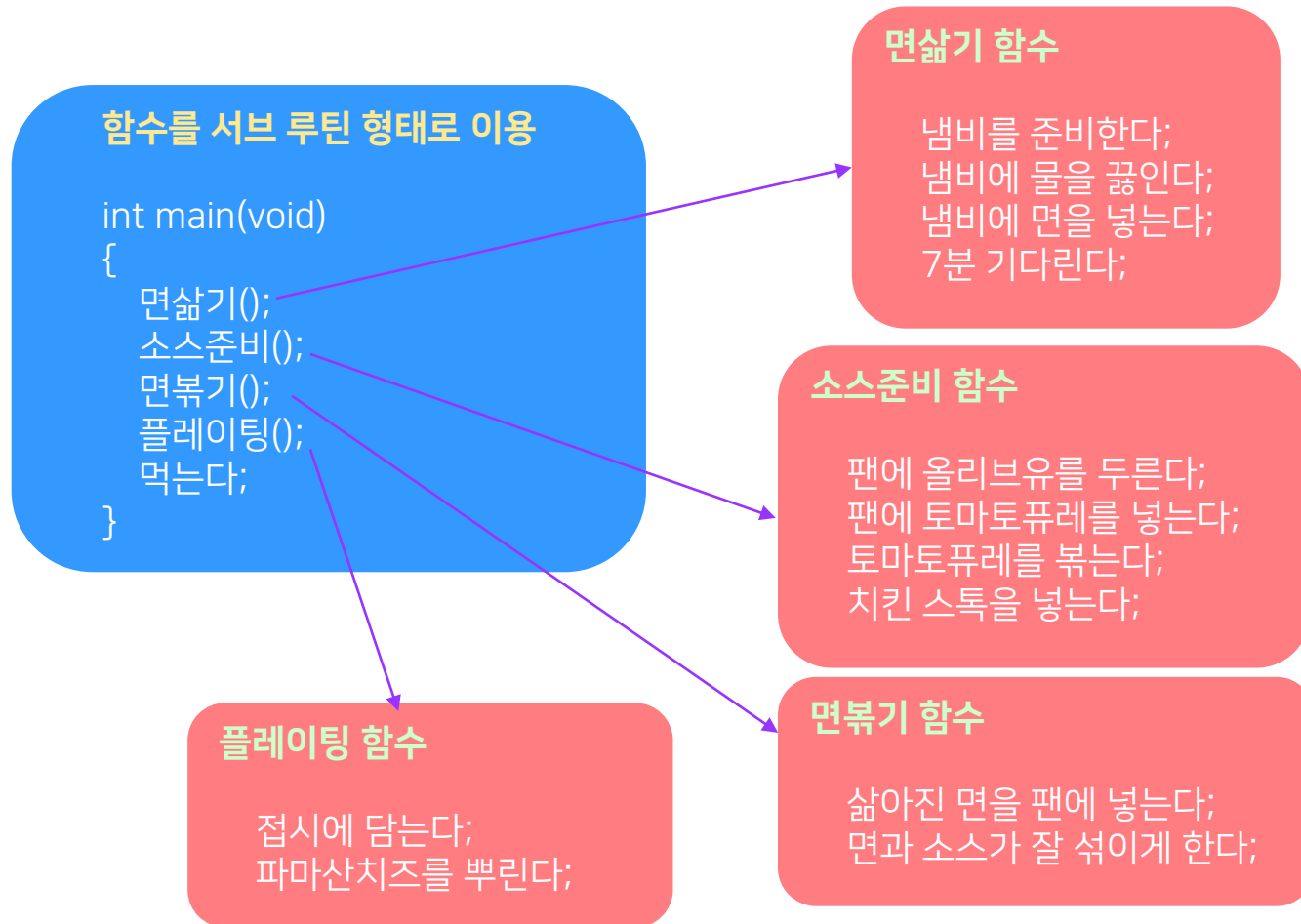
- 서브 루틴 형태로 이용 - 스파게티 요리 프로그램

기존 코드

```
int main(void)
{
    냄비를 준비한다;
    냄비에 물을 끓인다;
    냄비에 면을 넣는다;
    7분 기다린다;
    팬에 올리브유를 두른다;
    팬에 토마토피레를 넣는다;
    토마토피레를 볶는다;
    치킨 스톡을 넣는다;
    삶아진 면을 팬에 넣는다;
    면과 소스가 잘 섞이게 한다;
    접시에 담는다;
    파마산치즈를 뿌린다;
    먹는다;
}
```

1. 함수의 의미

■ 서브 루틴 형태로 이용 - 스파게티 요리 프로그램



1. 함수의 의미

- 서브 루틴 형태로 이용 - 스파게티 요리 프로그램
 - 코드 가독성
 - 모듈성
 - Scoping
 - Self-contained
 - 코드 재사용
 - 유지보수성
 - 등등

2. 함수의 사용

❖ 함수의 구조

Syntax: 함수 정의

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

반환형 함수 이름

매개 변수(현재는 없다)

함수 몸체

❖ 반환형

- 반환형은 함수가 처리를 종료한 후에 호출한 곳으로 반환하는 데이터의 유형

❖ 함수 이름

- 함수 이름은 식별자에 대한 규칙만 따른다면 어떤 이름이라도 가능
- 함수의 기능을 암시하는 (동사+명사)를 사용하면 좋음

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


2. 함수의 사용

- 복잡한 코드를 함수로 추출

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

int main(void)
{
    int from, to;
    bool is_prime;
    scanf("%d %d", &from, &to);

    for (int i = from; i <= to; i++) {
        is_prime = true;
        for (int j = 2; j < i; j++) {
            if (i % j == 0) {
                is_prime = false;
                break;
            }
        }
        if (is_prime) {
            printf("%d ", i);
        }
    }

    return 0;
}
```

2. 함수의 사용

- 복잡한 코드를 함수로 추출

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

bool is_prime(int num);

int main(void)
{
    int from, to;
    scanf("%d %d", &from, &to);

    for (int i = from; i <= to; i++) {
        if (is_prime(i)) {
            printf("%d ", i);
        }
    }

    return 0;
}
```

```
bool is_prime(int num)
{
    for (int i = 2; i < num; i++) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}
```

2. 함수의 사용

■ 실습1

- 순열 계산 코드를 함수로 추출하여
- 중복되는 코드를 줄여보자

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

int main(void)
{
    int n1, r1, n2, r2, result1 = 1, result2 = 1;

    printf("n1, r1을 입력해 주세요\n>>> ");
    scanf("%d %d", &n1, &r1);

    printf("n2, r2을 입력해 주세요\n>>> ");
    scanf("%d %d", &n2, &r2);

    for (int i = 0; i <= r1 - 1; i++) {
        result1 *= n1 - i;
    }

    for (int i = 0; i <= r2 - 1; i++) {
        result2 *= n2 - i;
    }

    char sign;
    if (result1 > result2) {
        sign = '>';
    } else if (result1 < result2) {
        sign = '<';
    } else {
        sign = '=';
    }

    printf("%d P %d %c %d P %d\n", n1, r1, sign, n2, r2);
    return 0;
}
```

2. 함수의 사용

- 라이브러리 함수 - math.h
- <https://www.ibm.com/docs/ko/i/7.3?topic=extensions-standard-c-library-functions-table-by-name>

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}

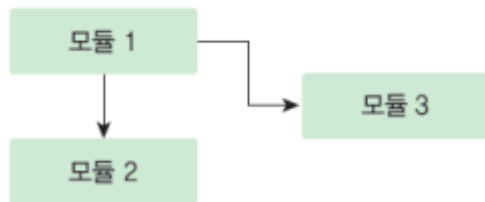
2. 함수의 사용

■ 실습2

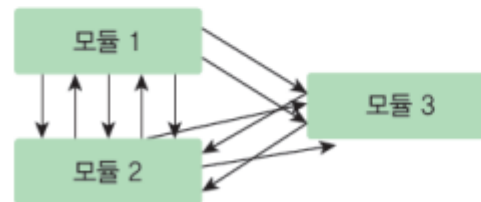
- 사용자로부터 double 타입 x 를 입력 받아 아래 식들을 계산한다.
 - $\cos(x) + \sin(x)$
 - $|-100 + \cos(x) - \sqrt{\sin(x)}|$

2. 함수의 사용

- ❖ 모듈이란 프로그램을 구성하는 시스템을 기능 단위로 독립적인 부분으로 분리한 것
 - 하나 이상의 논리적인 기능을 수행하는 명령들의 집합
- ❖ 모듈은 언어에 따라 함수, 프로시저(procedure), 서브루틴(subroutine), 메소드(method) 등으로 칭함
- ❖ 프로그램을 모듈로 구성하는 방법을 모듈화(modulization)하고 함
- ❖ 높은 응집도(cohesion)와 낮은 결합도(coupling)
 - 모듈 내에서는 최대의 상호 작용이 있어야 하고 모듈 사이에는 최소의 상호 작용만 존재해야 함
 - 만약 모듈과 모듈 사이의 연결이 복잡하다면 모듈화가 잘못된 것임



(a) 좋은 모듈화



(b) 나쁜 모듈화

3. 변수

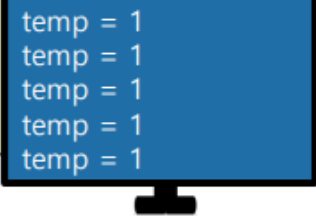
■ 지역변수

- ❖ 다른 함수나 다른 블록에서 이름이 같은 지역 변수 사용 가능
- ❖ 생존 시간: 지역변수는 선언된 블록이 끝나면 자동 소멸
 - 함수 호출 시 시스템 스택(stack)을 이용한 메모리 관리

```
#include <stdio.h>

int main(void)
{
    int i;

    for(i = 0; i < 5; i++)
    {
        int temp = 1;
        printf("temp = %d\n", temp);
        temp++;
    }
    return 0;
}
```



```
temp = 1
temp = 1
temp = 1
temp = 1
temp = 1
```

3. 변수

■ 전역변수

- ❖ 전역 변수(global variable)는 함수 외부에서 선언되는 변수
- ❖ 전역 변수의 범위는 소스 파일 전체임
- ❖ 생존 시간: 프로그램 시작과 동시에 접근 가능하며 종료되면 끝남

```
#include<stdio.h>
int A;
int B;
int add()
{
    return A + B;
}

int main()
{
    int answer;
    A = 5;
    B = 7;
    answer = add();
    printf(" % d + % d = % d\n", A, B, answer);
    return 0;
}
```

5 + 7 = 12

3. 변수

■ 변수 접근

- 함수를 이용한다면
- 각각의 함수마다 변수 공간을
- 분리하여 사용할 수 있다.
- 예시)
- A기능을 하는 함수 func_A에서는 A 기능 구현에 필요한 변수에만 접근할 수 있다.

```
#include <stdio.h>

int global = 10;

void foo(void);

int main(void)
{
    int local1 = 5;

    local1 = 7;
    local2 = 10.5; // local2 변수에 접근할 수 없다.

    global = 50; // global 변수에는 접근할 수 있다.
}

void foo(void)
{
    double local2 = 5.14;

    local1 = 10; // local1에 접근할 수 없다.
    local2 = 8.4; // local2에 접근할 수 있다.

    global = 78; // global 변수에는 접근할 수 있다.
}
```

3. 변수

■ Static 변수

```
#include <stdio.h>

int accumulate(int num);

int main(void) {
    accumulate(5);
    int sum = accumulate(7);

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

    return 0;
}

int accumulate(int num) {
    int val = 0;

    val += num;

    return val;
}
```

실행결과: 7

```
#include <stdio.h>

int accumulate_static(int num);

int main(void) {
    accumulate_static(5);
    int sum = accumulate_static(7);

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

    return 0;
}

int accumulate_static(int num) {
    static int val = 0;

    val += num;

    return val;
}
```

실행결과: 12

3. 변수

■ 실습3 - 로그인 함수

- 함수 **login(char pw)**가 있다.
- login 함수는 매개변수 pw에 비밀번호를 담아 호출한다.
- 만약 비밀번호가 틀리다면 "비밀번호 오류 (n회 남음)"을 출력한다.
 - 비밀번호: 'K'
 - n: 남은 시도 횟수. 총 5번의 기회가 주어진다.
- 만약 비밀번호를 올바르게 입력했다면 "로그인 성공 " 을 출력하고 프로그램을 종료 한다.
- 만약 비밀번호를 5번 다 틀렸다면, 로그인 함수를 호출 했을 때 계속해서 "시도 불가 " 를 출력하고 프로그램을 종료 한다.
- **printf는 login 함수 내에 위치 해야함**

```
비밀번호 입력: d  
비밀번호가 틀립니다. (5회 남음)
```

```
비밀번호 입력: c  
비밀번호가 틀립니다. (4회 남음)
```

```
비밀번호 입력: g  
비밀번호가 틀립니다. (3회 남음)
```

```
비밀번호 입력: K  
로그인 성공
```

```
D:\git\WC-TAWC Programming\Practice - 실습  
드: 0개).  
이 창을 닫으려면 아무 키나 누르세요...
```

4. 순환 호출

■ 순환/재귀 호출

- ❖ 순환/재귀(recursion)은 알고리즘이나 함수가 자기 자신을 호출하여 문제를 해결하는 기법
- ❖ 순환/재귀 호출(recursion call)은 함수 내부에서 함수가 자기 자신을 또다시 호출하는 행위를 의미함 => 순환/재귀 함수(recursion function)
- ❖ 순환 호출은 많은 문제들을 해결하는 독특한 개념적인 프레임워크 제공
- ❖ 예시: 팩토리얼 계산

$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)! & n \geq 1 \end{cases}$$

```
int factorial(int n)
{
    if( n <= 1 ) return(1);
    else return (n * factorial(n-1) );
}
```

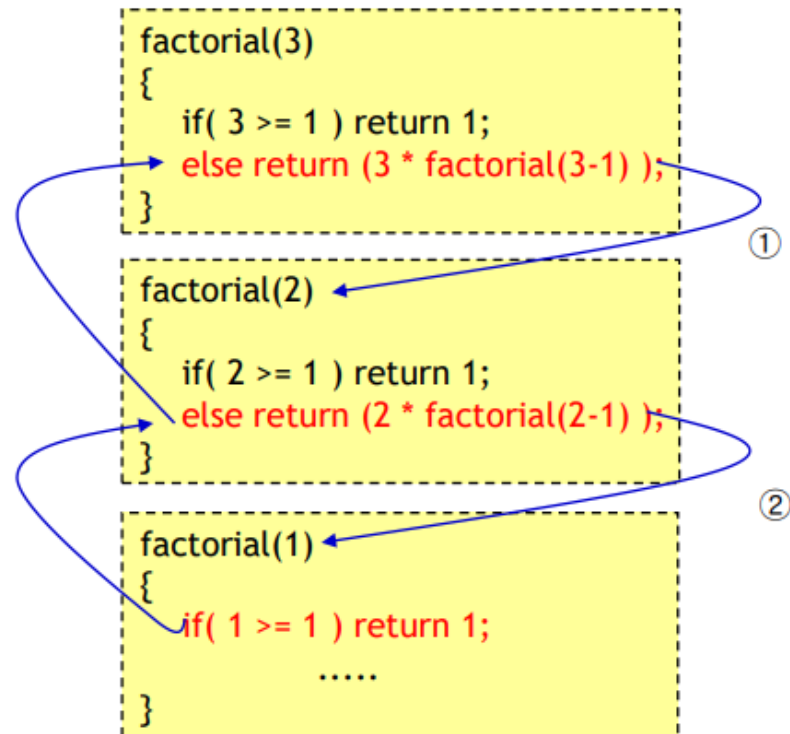
4. 순환 호출

■ 순환/재귀 호출

❖ 팩토리얼(factorial) 계산

- 팩토리얼의 호출 순서

$$\begin{aligned}\text{factorial}(3) &= 3 * \text{factorial}(2) \\ &= 3 * 2 * \text{factorial}(1) \\ &= 3 * 2 * 1 \\ &= 3 * 2 \\ &= 6\end{aligned}$$



4. 순환 호출

- 실습4 - 피보나치 수열
 - 피보나치 수열을 순환 호출 방식으로 개발해보자

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

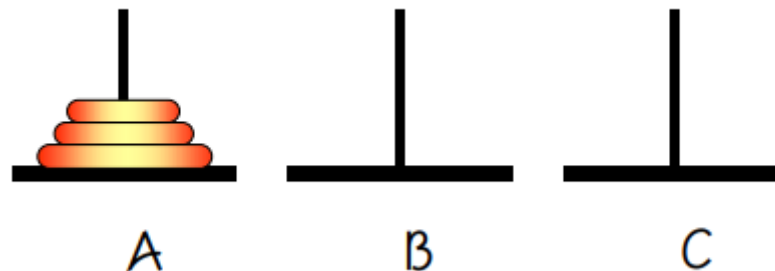
4. 순환 호출

■ 실습5 - 하노이의 탑

■ 하노이의 탑 문제

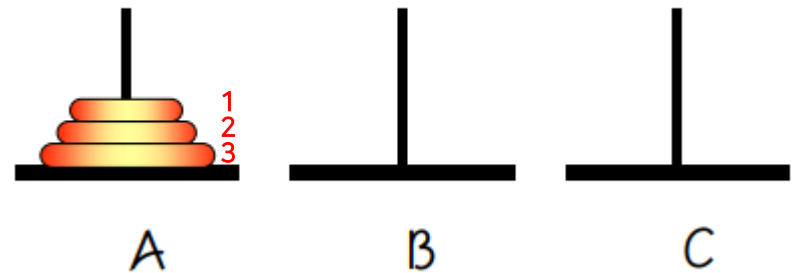
- 문제는 막대 A에 쌓여있는 원판 3개를 막대 C로 옮기는 것이다.
- 단 다음의 조건을 지켜야 한다.
 - 한번에 하나의 원판만 이동할 수 있다.
 - 맨 위에 있는 원판만 이동할 수 있다.
 - 크기가 작은 원판 위에 큰 원판이 쌓일 수 없다.
 - 중간막대를 임시적으로 이용할 수 있으나, 앞의 조건들을 지켜야 한다.

- 팩토리얼, 피보나치 수열 문제에서 처럼 하노이의 탑 문제를 각 단계별로 쪼개어보자



4. 순환 호출

- 실습5 - 하노이의 탑
 - 원판 3, 2, 1을 A에서 C로 옮긴다.
 - 원판 2, 1을 A에서 B로 옮긴다.
 - 원판 1을 A에서 C로 옮긴다.
 - 원판 2를 A에서 B로 옮긴다.
 - 원판 1을 C에서 B로 옮긴다.
 - 원판 3을 A에서 C로 옮긴다.
 - 원판 2, 1을 B에서 C로 옮긴다.
 - 원판 1을 B에서 A로 옮긴다.
 - 원판 2를 B에서 C로 옮긴다.
 - 원판 1을 A에서 C로 옮긴다.



4. 순환 호출

- 실습5 - 하노이의 탑
 - 일반화
 - 출발막대를 src, 목적막대를 dest, 예비막대를 aux
 - $n, n-1, n-2, \dots$ src to dest
 - $n-1, n-2, \dots$ src to aux
 - n src to dest
 - $n-1, n-2, \dots$ aux to dest
 - 1 src to dest
 - 1 src to dest