

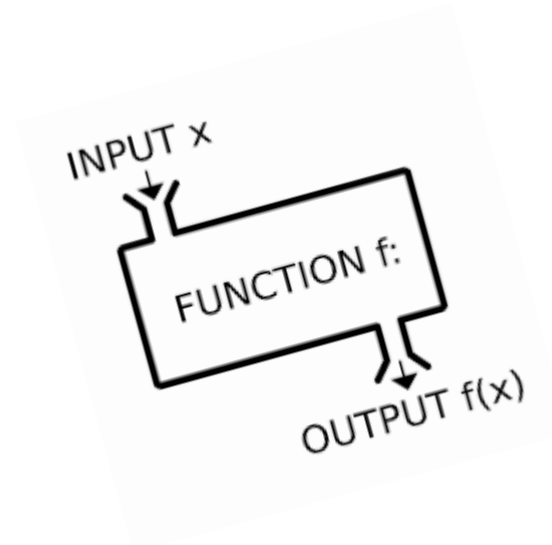
**Robotics**



대경혁신인재양성프로젝트  
**HuStar**

# C프로그래밍2

### Function Declaration

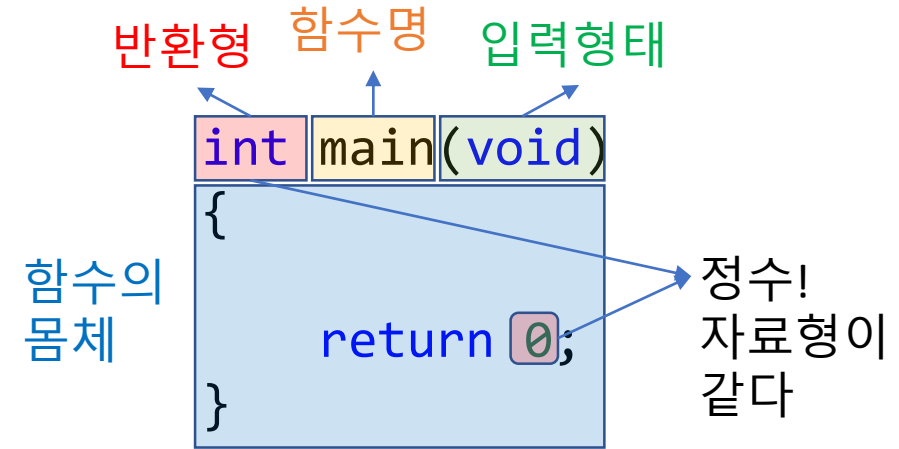


The diagram illustrates the components of a C++ function signature: `float findArea(float length, float width);`. Annotations with arrows identify each part:   
- **return type**: points to `float` at the start.   
- **function name**: points to `findArea`.   
- **parameter type**: points to `float` before `length`.   
- **parameter name**: points to `length`.   
- **parameter type**: points to `float` before `width`.   
- **parameter name**: points to `width`.   
- **terminating semicolon**: points to the semicolon at the end.

# 함수(function)

- 함수

- 여러 명령이 있는 작은 프로그램 단위
- C는 하나 이상의 함수로 이루어져 있음
- 구조적 프로그래밍(structured programming)
- 소프트웨어 재사용성 증가
  - 새로운 프로그램을 만들 경우 기존의 만들어 두었던 함수를 사용할 수 있음



- 함수는 반환형, 이름, 입력형태, 몸체로 구성
  - 반환형: 함수 종료 시 반환되는 데이터 형식
  - 이름: 함수를 호출할 때 사용하는 이름
  - 입력형태: 함수가 동작할 때 필요한 데이터(parameters)
  - 몸체: 함수가 동작하는 실제 내용

주의! 함수의 반환형과 return의 자료형이 같아야 한다.

# 함수의 종류

- Library 함수(library function)
  - 시스템에서 미리 정의된 함수
  - 예시) printf(), scanf(), getchar(), putchar() ...
- 사용자 정의 함수(user define function)
  - 프로그래머가 만든 함수
  - 함수를 사용하기 위한 단계
    - 함수 선언: 함수가 있다는 것을 알림
    - 함수 정의: 함수의 동작을 작성
    - 함수 호출: 함수의 사용

# 함수의 형태

- 입력형태와 반환형에 따라서 4가지로 나누어짐

Case 1: 반환 값 있음, 전달 인자 있음

```
int Add(int num1, int num2){  
    int result=num1+num2;  
    return result;  
}
```

Case 2: 반환 값 없음, 전달 인자 있음

```
void ShowAddress(int num){  
    printf("덧셈결과 출력: %d\n", num);  
}
```

Case 3: 반환 값 있음, 전달 인자 없음

```
int ReadNum(){  
    int num;  
    scanf("%d",&num);  
    return num;  
}
```

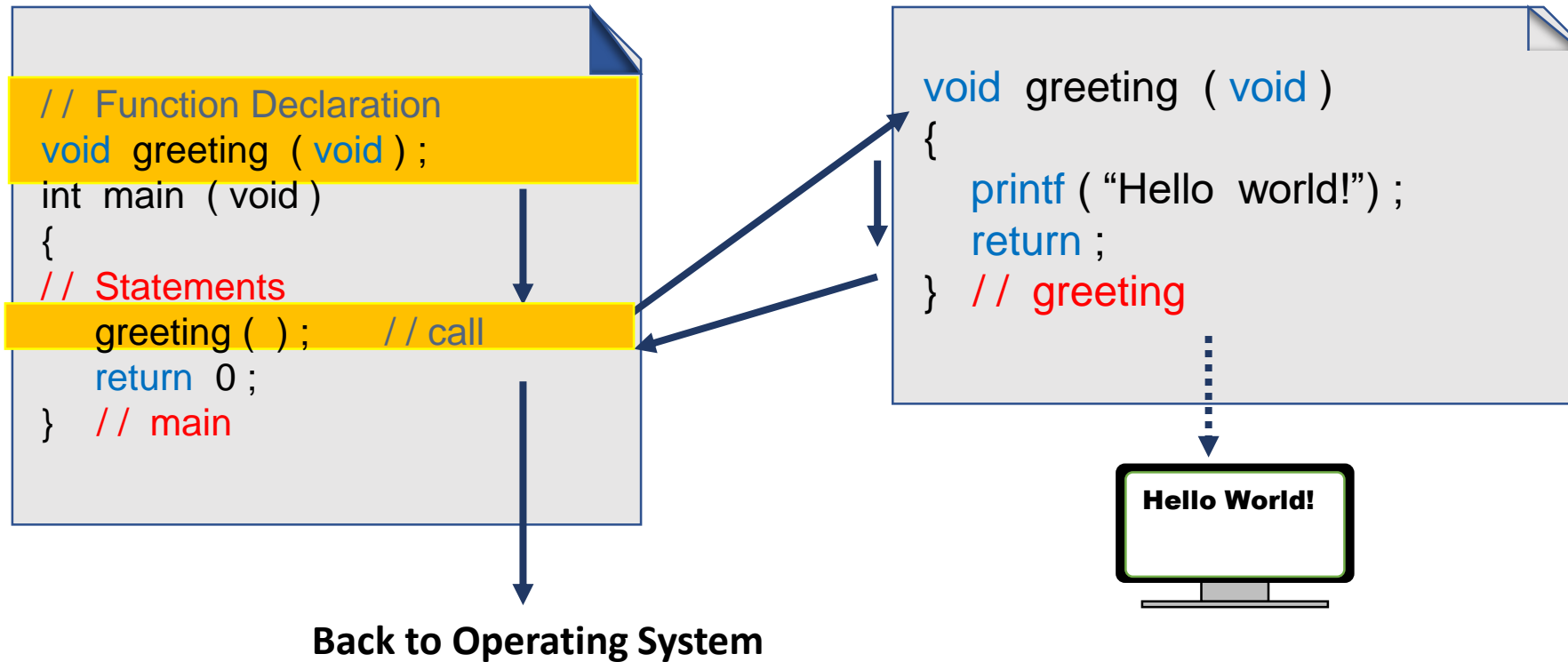
Return 이 없는 경우 반드시 void라고 작성

Case 4: 반환 값 없음, 전달 인자 없음

```
void HowToUseThisProg(void){  
    printf("두 개의 정수 입력! 덧셈 결과 출력\n");  
    printf("두 개의 정수를 입력하세요. \n");  
}
```

입력형태(전달인자)가 없는 경우 void 생략 가능

# 함수의 동작



# 함수 사용 예시

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

```
int multiply(int num1, int num2);
```

함수 선언:  
컴파일러에게 함수가 있다는 것을 알려줌

```
int main(){  
    int multiplier, multiplicand, product;
```

```
    printf("Enter two integers: ");  
    scanf("%d%d", &multiplier, &multiplicand);
```

```
    product = multiply (multiplier, multiplicand);
```

함수 호출:  
함수를 사용

```
    printf("product of %d & %d is %d\n", multiplier, multiplicand, product);  
    return 0;
```

```
} //main
```

```
int multiply ( int num1, int num2){  
    return (num1 * num2);  
} //multiply
```

함수 정의:  
함수가 어떤 동작을 할지 정의  
실제 함수의 몸체

# 함수 사용 예시2

```
#include <stdio.h>
int Add(int num1, int num2){ 함수 정의
    return num1+num2;
}

int main(){
    int result;
    result = Add(3,4); 함수 호출
    printf("덧셈결과1: %d \n", result);
    result = Add(5,8); 함수 호출
    printf("덧셈결과2: %d \n", result);
    return 0;
}
```

함수의 선언이 반드시 필요한 것은 아님  
함수의 정의가 함수 호출 전에 있다면, 호출 가능



# 함수-return의 활용

- Return의 의미
  - 함수의 종료
  - 값의 반환
- Return은 반드시 하나의 값으로 이루어 져야 함

return 0;

return x+a+12; => 수식이 return에 있는 경우 수식의 결과를 되돌려 줌

return 0 ;	상수
return status ;	변수
return i > j ? i : j ;	조건(삼항) 연산자
return ;	Void type에 사용 반드시 필요하지는 않음

```
void Noreturntype(int num){  
    if(num<0)  
        return ;  
}
```

함수를 종료의 의미만 가진 return 문

# 함수 예시

- 하나의 함수 내에서 둘 이상의 return문

```
#include <stdio.h>
int numberCompare(int num1, int num2);

int main(){
    printf("3과 4중에서 큰수는 %d 이다.\n", numberCompare(3,4));
    printf("7과 2중에서 큰수는 %d 이다.\n", numberCompare(7,2));
}

int numberCompare(int num1, int num2){
    if(num1>num2)
        return num1;
    else
        return num2;    두 값을 비교하여 큰 값을 가진 변수 return
}
```

3과 4중에서 큰수는 4 이다.  
7과 2중에서 큰수는 7 이다.

# 함수의 기본 자료형

- 함수의 반환형의 기본 자료형
  - 정수형(int)
  - 자료형이 없는 함수의 경우 int로 판단

(C89)Average 함수는  
자료형이 없으므로 int로  
판단

- 즉, 반환형이 int인  
경우 생략 가능

(C99)에서는 함수의  
반환형을 생략하는 경우  
illegal(warning or error)

```
average( int a, int b) {  
    int sum;  
  
    sum = a + b;  
    return sum /2 ;    //int/int produces int result.  
}
```

# 다중 인자 함수(Multi parameter)

- 인자(parameter)
  - 함수의 정의에서 입력형태에 나열되는 변수
- 전달 값(argument)
  - 함수를 호출할 때, 전달 혹은 입력되는 실제 값

```
#include <stdio.h>
int main() {
    int a ;

    printf("fun(5) = %d\n", a=fun(5));
    printf("a = %d\n", a );
}
```

argument

```
int fun (int a) {
    a = a + 3 ;
    return a ;
}
```

parameter

# 다중 인자 함수(multi parameter)

- 다중 인자 함수
  - 2개 이상의 인자를 사용한 함수

```
double average (int x, int y)
```

Parameter를 2개  
사용하는 함수

```
{  
    double sum ;  
    sum = x + y ;  
    return ( sum / 2 ) ;  
} // average
```

# 다중 인자 함수 예시

```
#include <stdio.h>
double calc(int num1, int num2, char oper);

int main(){
    int a, b;
    char opt;

    printf("연산입력(3+2):");
    scanf("%d%c%d",&a,&opt,&b);

    printf("%d %c %d = %lf",a,opt,b,calc(a,b,opt));

    return 0;
}

double calc(int num1, int num2, char oper){
    double result;
    switch(oper){
        case '+': result=num1+num2; break;
        case '-': result=num1-num2; break;
        case '*': result=num1*num2; break;
        case '/': result=num1/num2; break;
        case '%': result=num1%num2; break;
        default: printf("error input\n"); result=0.0;
    }
    return result;
}
```

실행결과

연산입력(3+2):6\*4  
6 \* 4 = 24.000000

# 함수 호출

- 함수 호출 형식
  - 호출하는 함수가 되돌려 주는 값이 있는 경우
    - 변수 = 함수();, 함수(함수);
  - 호출하는 함수가 되돌려 주는 값이 없는 경우
    - 함수();

```
print_count(i);           //void function
print_prn();              //void, no parameter function
avg = average(x, y);      //return valued function
printf("The...%f \n", average(x, y) ); // return valued function
```

# 함수 호출 예시

- 함수 호출에서 argument는 상수, 변수, 함수, 수식이 올 수 있음

**multiply ( 6, 7 )**

**multiply ( 6, b )**

**multiply (multiply ( a, b ), 7 )**

**multiply ( a, 7 )**

**multiply ( a + 6, 7 )**

**multiply ( ... , ... )**

expression

expression



# 함수 호출 오류 찾기

- 아래 코드에서 오류를 찾으시오.

```
/* wrong example */
#include <stdio.h>
int fun ( int num );

int main() {
    int a = 5 ;

    printf("fun(a) = %d\n", fun( ) );
    printf("a = %d\n", a );
}

void fun ( int a ) {
    a = a + 3 ;
    return a ;
}
```

```
/* wrong example */
#include <stdio.h>
float average (float a, float b);

int main() {
    float x, y, z ;

    printf("Enter two numbers ;");
    scanf("%f%f", &x, &y);
    printf("%f\n", average(x, y) );
}

float average ( int a, int b ) {
    return (a + b) /2 ;
}
```

# 함수 실습

- 3개의 정수를 받아 평균을 계산하는 avr3()와 main() 구현
- 평균 출력 시 소수점 2자리까지 출력되어야 함

```
input 3 integers(3 2 4):11 2 3
numbers 11, 2, 3 are average 5.00
```

# 함수 실습2

- 다음 수식의 결과를 계산하는 프로그램 작성
- x는 사용자 입력

$$3x^5+2x^4-5x^3-x^2+7x-6$$

```
Enter value for x: 3  
Polynomial value: 762
```

# 함수 실습3

- 문자 2개를 입력 받고 오름차순으로 출력하는 프로그램
- 문자 크기를 비교하는 함수를 구현

```
input capital letter:ga  
a g
```

# 함수에서 인수 전달

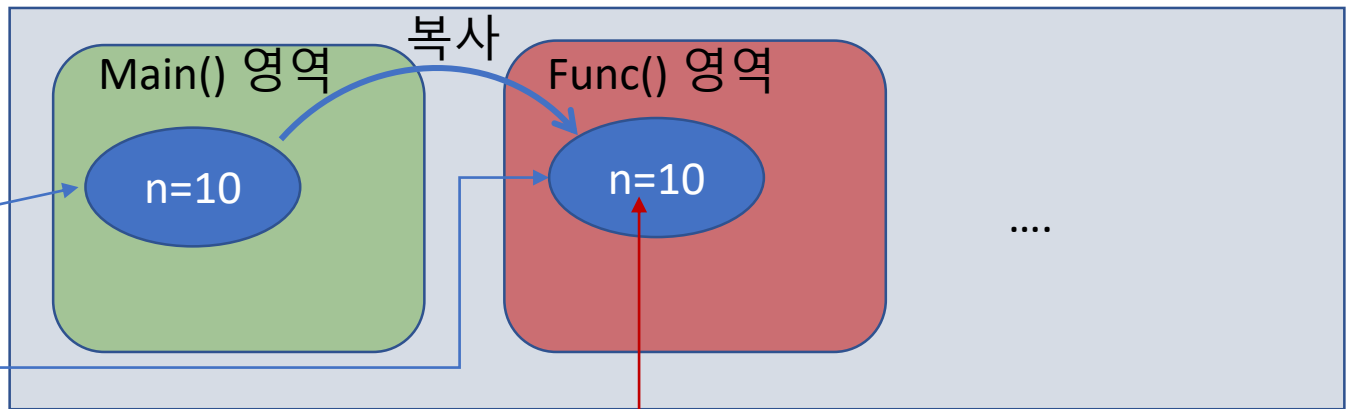
- Call by value

- 함수에서 인수를 전달하는 일반적인 방법
- 인수를 복사하여 전달 해당 함수에 전달하기 때문에
- 함수 호출에 사용되었던 인수들은 함수 내에서 변화하지 않음

```
#include <stdio.h>
void func(int n){
    n=20;
}

int main(){
    int n=10;
    func(n);
    printf("%d",n);
}
```

컴퓨터 메모리



func()의 n=20으로 수정 하여도  
main()의 n의 값은 변화가 없음

# Call by value 예시

- 두 값을 변경하는 swap 프로그램

```
#include <stdio.h>

void swap(int a, int b){
    int temp;

    temp = a;
    a = b;
    b = temp;
    printf("In swap function:%d,%d\n",a,b);
}

int main(){
    int a=10, b=20;

    printf("swap 전 : %d %d\n", a, b);

    swap(a, b);

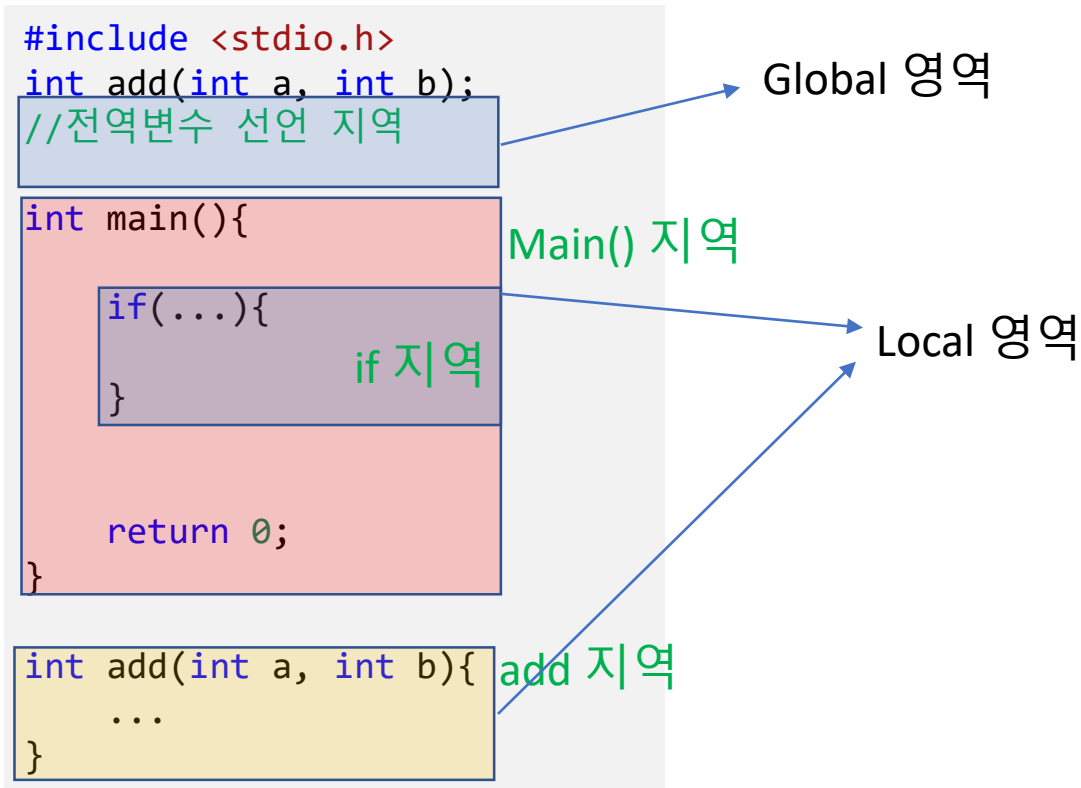
    printf("swap 후 : %d %d\n", a, b);

    return 0;
}
```

```
swap 전 : 10 20
In swap function:20,10
swap 후 : 10 20
```

# 지역(local)/전역(global) 변수

- 변수의 경우 선언 위치에 따라서 지역변수, 전역변수로 나누어짐
  - 지역변수(local):** 함수 내에만 존재 및 접근 가능한 지역변수
  - 전역변수(global):** 프로그램의 시작부터 종료까지 메모리에 유지되는 변수



# 지역변수(local variable)

- 지금까지 지역변수만을 사용했음
- **지역변수의 특징**
  - 해당 영역 내에서만 접근 및 수정 가능
  - 지역은 {}로 생성 되는 영역
  - 지역변수는 해당 지역을 벗어나면 자동 소멸
  - 지역변수는 선언된 지역 내에서만 유효

```
#include <stdio.h>

int main(){
    int cnt;

    for(cnt=0; cnt<3; cnt++){
        int num=0;
        num++;
        printf("%d번째 반복, 지역변수 num은 %d. \n", cnt+1, num);
    }

    if(cnt==3){
        int num=7;
        num++;
        printf("if문 내에 존재하는 지역변수 num은 %d. \n", num);
    }

    return 0;
}
```



# 전역변수(global variable)

- 함수의 외부에서 선언되는 변수
- 전역변수의 특징
  - 프로그램의 시작과 동시에 메모리 공간에 할당되어 종료 시까지 존재
  - 프로그램 전체 영역 어디서든 접근 가능
  - 전역변수의 경우 자동으로 0으로 초기화

```
#include <stdio.h>
void Add(int val);
int num = 0; //global variable, 0으로 초기화

int main(){
    printf("num: %d \n", num);
    Add(3);
    printf("num:%d \n", num);
    num++;
    printf("num:%d \n", num);
    return 0;
}

void Add(int val){
    num +=val; //전역변수 num에 val만큼 증가
}
```

# 전역/지역 변수

- 동일한 이름의 전역/지역 변수 있을 경우
  - 해당 지역 내에서는 **지역변수를 우선하여 접근**

```
#include <stdio.h>
int add(int val);
int num=1;

int main(){
    int num=5;

    printf("num: %d \n", add(3));
    printf("num: %d \n", num+9);
    return 0;
}

int add(int val){
    int num=9;
    num += val;
    return num;
}
```

전역변수 ←

지역변수 ←

전역변수를 사용을 최대한 신중하게 하자

-전역변수를 많이 사용하는 경우  
프로그램이 복잡해지는 문제가 있음

```
num: 12
num: 14
```

# Static 지역변수

- 지역변수 선언에 static 키워드를 추가하여 전역변수의 일부 특징을 추가
  - Static이 추가된 변수
    - 선언된 함수 내에서만 접근이 가능 (지역변수 특징)
    - 1회 초기화되고 프로그램 종료 까지 메모리에 존재 (전역변수 특징)

```
#include <stdio.h>

void simpleFunc(void){
    static int num1=0; //초기화 하지 않으면 자동으로 0초기화
    int num2=0;
    num1++, num2++;
    printf("static: %d, local: %d \n", num1, num2);
}

int main(){
    for(int i=0;i<3;i++)
        simpleFunc();

    return 0;
}
```

Static으로 선언된 지역변수는 전역변수와 동일한 시기에 할당되고 소멸됨.  
단, 선언된 함수 내에서만 접근 가능

```
static: 1, local: 1
static: 2, local: 1
static: 3, local: 1
```

# Static 지역변수- 실습

- 사용자의 입력을 누적하여 합계 출력 프로그램
- Total 전역변수를 static지역변수로 대체 할것, main함수 변경 불가

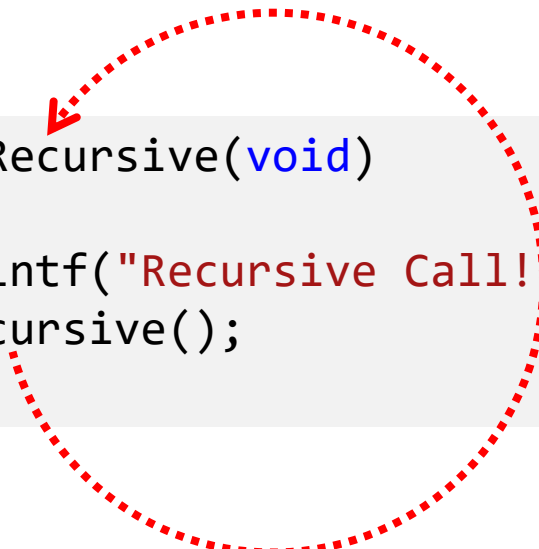
```
#include <stdio.h>
int total=0;

int addToTotal(int num){
    total+=num;
    return total;
}
int main(void){
    int num, i;
    for(i=0; i<3;i++){
        printf("input%d:",i+1);
        scanf("%d",&num);
        printf("total: %d \n", addToTotal(num));
    }
    return 0;
}
```

```
input1:3
total: 3
input2:3
total: 6
input3:3
total: 9
```

# 함수를 이용한 반복-재귀(recursion)

- 재귀함수(recursive function)
- 반복을 하는 방법
  - Iteration: for, while문과 같이 반복문을 이용한 반복
  - Recursion: 함수의 자기 호출을 이용한 반복



```
void Recursive(void)
{
    printf("Recursive Call! \n");
    Recursive();
}
```

# 재귀함수(recursive function)

- 재귀함수
  - 자기자신을 호출하는 함수
  - 호출을 종료(탈출!)하는 시점이 반드시 필요
  - Recursive 함수가 다시 recursive 함수를 호출하면 recursive 함수의 복사본을 만들어서 실행

```
#include <stdio.h>
void recursive(int num){
    if(num<=0) //재귀탈출조건
        return; //탈출
    printf("Recursive call! %d \n", num);
    recursive(num-1);
}

int main(void){
    recursive(3);
    return 0;
}
```

```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```

# 재귀를 이용한 factorial

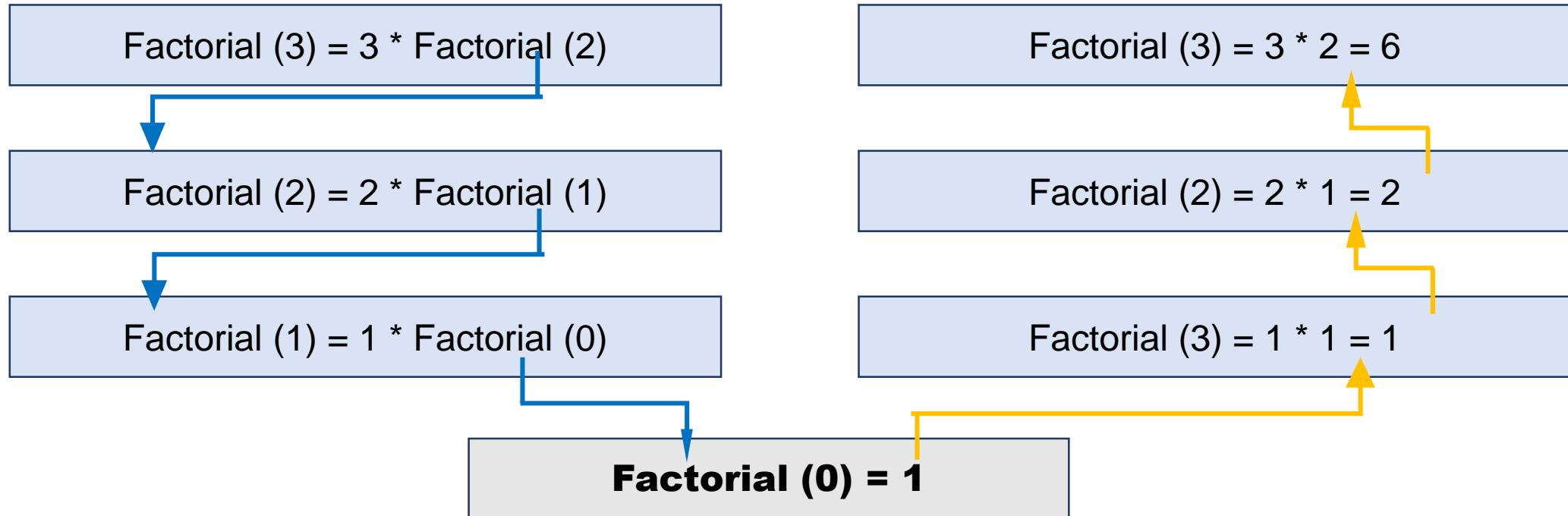
- Factorial:  $3!=3*2*1$ 을 의미

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) * \dots * 3 * 2 * 1 & \text{if } n \neq 0 \end{cases}$$

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * factorial(n-1) & \text{if } n \neq 0 \end{cases}$$

# 재귀를 이용한 factorial

- Factorial (3) Recursively





# 재귀를 이용한 factorial

- Iteration과 recursion

```
/* Calculate the factorial of a number using a loop */
```

```
int factorial(int n){  
    int factn = 1, i;           //local declarations  
  
    for (i=1; i<=n; i++)  
        factn *= i;  
    return factn;  
  
} //factorial
```

```
/* Calculate the factorial of a number using recursion */
```

```
int factorial(int n){  
    if (n ==0)  
        return 1;  
    else  
        return (n* factorial (n-1));  
  
} //factorial
```

# 재귀를 이용한 factorial

```
#include <stdio.h>

int factorial(int n){

    if (n == 0)
        return 1;
    else
        return (n* factorial (n-1));

}

int main(){
    printf("1! = %d \n",factorial(1));
    printf("2! = %d \n",factorial(2));
    printf("3! = %d \n",factorial(3));
    printf("4! = %d \n",factorial(4));
    printf("9! = %d \n",factorial(9));
    return 0;
}
```

1! = 1  
2! = 2  
3! = 6  
4! = 24  
9! = 362880

# 재귀함수 실습

- 피보나치 수열 구하기
  - 1 1 2 3 5 8 13 ...
  - N항은 n-1항과 n-2항을 더한 것
  - $f_n = f_{n-1} + f_{n-2} \quad n \geq 3$
  - $f_1 = f_2 = 1 \quad n = 1, 2$

1:	1
2:	1
3:	2
4:	3
5:	5
6:	8
7:	13
8:	21
9:	34
10:	55
11:	89
12:	144
13:	233
14:	377
15:	610
16:	987
17:	1597
18:	2584
19:	4181