

---

# C 프로그래밍 및 실습

## 8. 함수

세종대학교

---

- 1) 함수란?
- 2) 함수 정의
- 3) 함수 호출과 반환
- 4) 지역변수와 전역변수
- 5) 배열 인자
- 6) 함수와 라이브러리

# 1) 함수란?

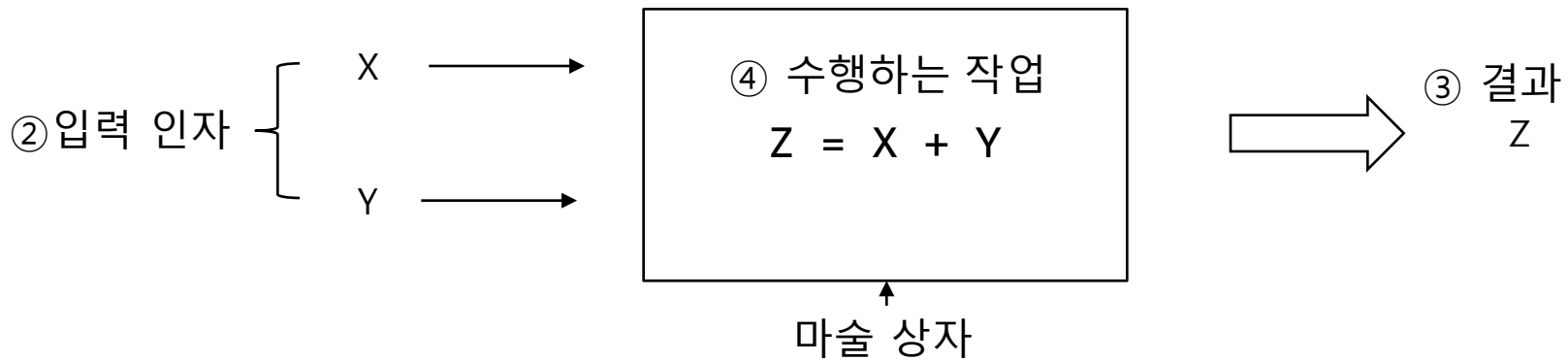
## ▪ C언어에서 함수

- 어떤 특정한 일을 수행하는 **독립적인** 단위
- 함수의 예 : printf(), scanf(), main()
- 수학에서의 함수 개념과 유사

## ▪ 수학에서 함수

- $f(x,y) = x + y$  : 두 수의 합을 구하는 함수
- 함수(마술상자) f에 x 와 y를 입력하면 두수의 합이 결과로 나옴

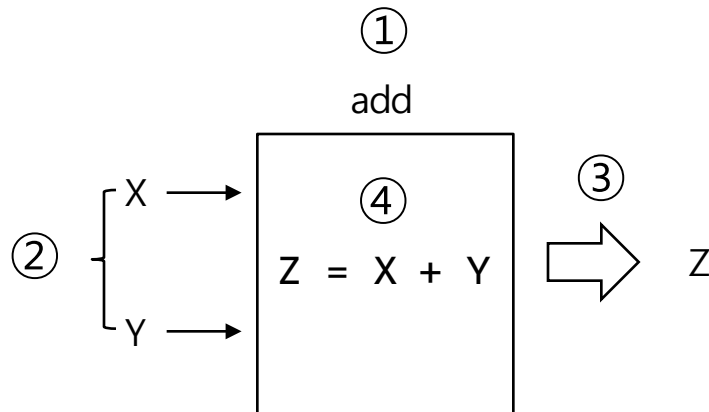
① 함수 이름  
f



- 함수 구성요소 4가지 : ①, ②, ③, ④

## 1) 함수란?

- 앞의 함수  $f(x,y)$ 를 C언어로 구현 하면?
  - 편의상, 함수 이름으로  $f$  대신  $add$ 를 사용



```
③ ① ②
int add(int x, int y)
{
    ④ { int z;
        z = x+y;
        return z;
    }
}
```

함수 정의부

- 구성요소 4가지가 전부 포함됨을 짐작할 수 있음

# 1) 함수란?

## ■ 함수는 함수 정의부와 함수 호출부로 구성

- **함수 정의부** : 함수를 구현하는 부분 (즉, 마술상자 내부 구현)
  - 함수 4가지 구성요소 전부 신경 써야 함
  - 함수 호출하는 부분의 상황 고려할 필요 없음
- **함수 호출부** : 함수의 기능을 이용하는 부분 (즉, 마술상자 이용)
  - 4가지 구성요소 중 함수이름, 입력인자, 결과(결과의 자료형)는 알아야 함
  - 함수 내부 구현을 전혀 신경 쓸 필요 없음

```
int add(int x, int y)
{
    int z;
    z = x+y;

    return z;
}
```

} 함수 정의부

```
int main()
{
    int c;
    c = add(3,4);
    printf("3 + 4 = %d\n", c);

    return 0;
}
```

← 함수 호출부

## 2) 함수 정의

---

- 함수

- printf(), scanf() 함수 처럼 **표준 함수**의 경우 이미 코드로 구현되어 컴파일러에 제공
- 사용자가 함수를 만들어 사용할 경우 함수의 내부 코드를 직접 작성해야 함

```
int add (int x, int y)
{
    int z;
    z = x+y;

    return z;
}
```

- 함수 정의하려면 네 가지 구성요소가 무엇인지 명세해야 함

## 2) 함수 정의

### ■ 함수 정의

```
  ③  ①      ②  
int  add (int x, int y)  
{  
    int z;  
    ④  z = x+y;  
    return z;  
}
```

#### ① 함수 이름

- 함수를 사용하기 위해서 이름 존재 해야 함
- 함수 이름은 변수 명을 선언할 때의 규칙과 동일
- 함수 이름은 함수가 하는 일을 표현하는 이름으로 선언

#### ② 함수 인자 or 매개 변수(parameter)

- 함수가 수행되기 위해 필요한 함수의 입력 데이터를 나타내는 변수들
- 인자가 다수인 경우 콤마(,)로 구분

※인자가 없더라도 소괄호는 반드시 필요

## 2) 함수 정의

### ▪ 함수 정의

```
  ③  ①      ②  
int add (int x, int y)  
{  
    int z;  
    ④  z = x+y;  
    return z;  
}
```

#### ③ 반환형(return type)

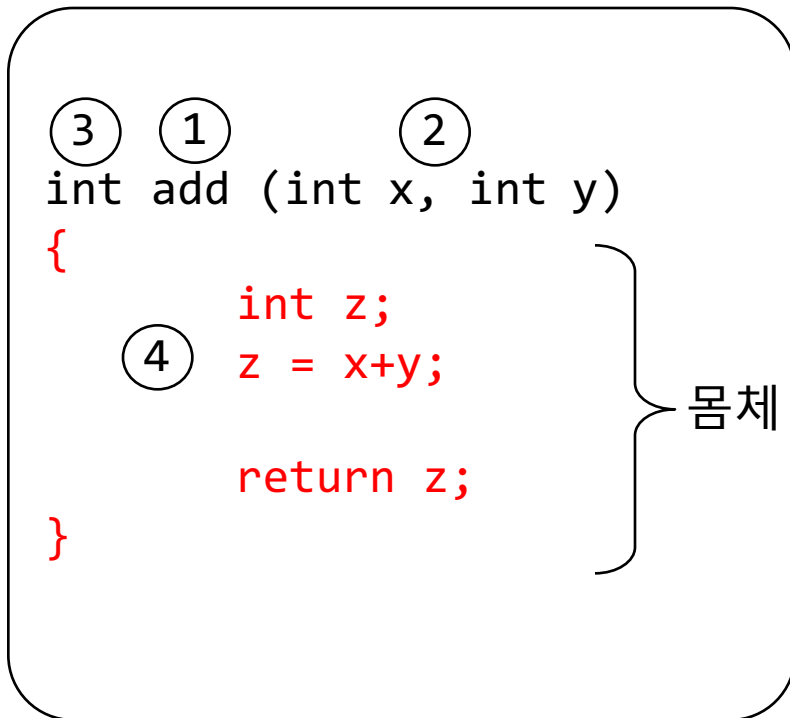
- 함수는 수행된 후의 결과를 반환 함
- 반환할 결과의 **자료형**을 명시
- 반환형은 함수 이름 앞에 적음
- 아무 결과도 반환 하지 않으려면  
자료형에 void를 씀

※ ①, ②, ③을 함수의 헤더(header)라 부르고, 함수의 형태를 명세하는 역할을 한다.



## 2) 함수 정의

### ■ 함수 정의



#### ④ 몸체(body)

- 함수가 수행해야 할 일을 헤더 다음에 중괄호 { }안에 명시하게 되는데, 이를 함수의 몸체(body)라고 함

- 함수는 제어 흐름에 따라 수행되다, 맨 마지막 문장까지 수행 되면 종료

- 함수의 결과를 반환하기 위해서는 return문을 사용

#### ※return문

- 함수 어느 곳에 위치해도 됨
- 함수 수행 도중 return문을 만나면 함수 종료
- 반환형이 void 인 경우 함수가 아무런 결과도 반환 하지 않는다는 의미이므로, return 문을 쓰지 않아도 됨

## 2) 함수 정의

- 일반적인 함수 정의 구문

```
반환형 함수이름 (인자선언1, 인자선언2, ...)  
{
```

```
    함수의 수행코드
```

```
    return 문;
```

```
}
```

인자 수는 원하는 대로  
정할 수 있다.

## 2) 함수 정의

### ■ 예제1

```
char next_char(char c, int num)
{
    char c1;

    . . .

    return c1;
}
```

#### 예제 1

1. 반환형 : char
2. 함수 이름 : next\_char
3. 함수 인자 : char형 변수 c, int형 변수 num

※ 반환형이 char형이기 때문에, return문이 없으면 컴파일 오류가 발생!!

### ■ 예제2

```
void print_heading( void )
{

    printf("\n===== \n");
    printf("    heading    ");
    printf("\n===== \n");

}
```

#### 예제 2

1. 반환형 : void
2. 함수 이름 : print\_heading
3. 함수 인자 : 없음

※ 반환형이 void형이기 때문에, return문을 쓰지 않아도 됨

※ ~~반환형 void를 생략하면 int형으로 간주됨~~

※ 인자는 없을 경우 void를 적거나 안 적어도 됨

## 2) 함수 정의

---

- **[실습1] 다음 조건을 만족하는 함수를 정의 하시오.**
  - max 함수
    - ✓ 함수 이름 : max
    - ✓ 인자 : int형 변수 a와 b
    - ✓ 반환형 : int형
    - ✓ a와 b 중 큰 값을 반환
- **[실습2] 다음 조건을 만족하는 함수를 정의 하시오.**
  - print\_characters 함수
    - ✓ 함수 이름 : print\_characters
    - ✓ 인자 : char형 변수 c와 int형 변수 n
    - ✓ 반환형 : void
    - ✓ 하나의 줄에 변수 c의 문자를 n개 출력

## 2) 함수 정의

---

- **[실습3] 다음 조건을 만족하는 함수를 정의 하시오.**
  - divide 함수
    - ✓ 함수 이름 : divide
    - ✓ 인자 : int형 변수 a와 b
    - ✓ 반환형: double
    - ✓ a를 b로 나눈 결과를 반환. 단, 실수 연산을 해야 함.
    - ✓ 예를 들어 3/2의 경우 1.5를 반환 (소수점 아래 한자리까지)
- **[실습4] 다음 조건을 만족하는 함수를 정의 하시오.**
  - add3 함수
    - ✓ 함수 이름 : add3
    - ✓ 반환형 : float
    - ✓ 인자 : float형 변수 a, b, c
    - ✓ a, b, c의 합을 반환

## 2) 함수 정의

---

- **[실습5] 다음 조건을 만족하는 함수를 정의 하시오.**
  - atoA 함수
    - ✓ 함수 이름 : atoA
    - ✓ 반환형 : char
    - ✓ 인자 : char형 변수 ch
    - ✓ 소문자인 ch를 대문자로 변환하여 반환

→ 이상 여기서 정의한 함수는 뒤에서 활용됨

### 3) 함수 호출과 반환

## ■ 함수 호출(사용)하는 방법

- 함수 이름을 쓰고, 소괄호 안에 함수 인자에 넣을 값을 차례로 적음
- `add(3, 4)`는 `add`함수의 첫 번째 인자가 3, 두 번째 인자가 4라는 것을 의미

```
int add(int x, int y)
{
    int z;

    z = x+y;

    return z;
}
```

```
int main()
{
    int c;

    c = add(3, 4); // 함수 호출부

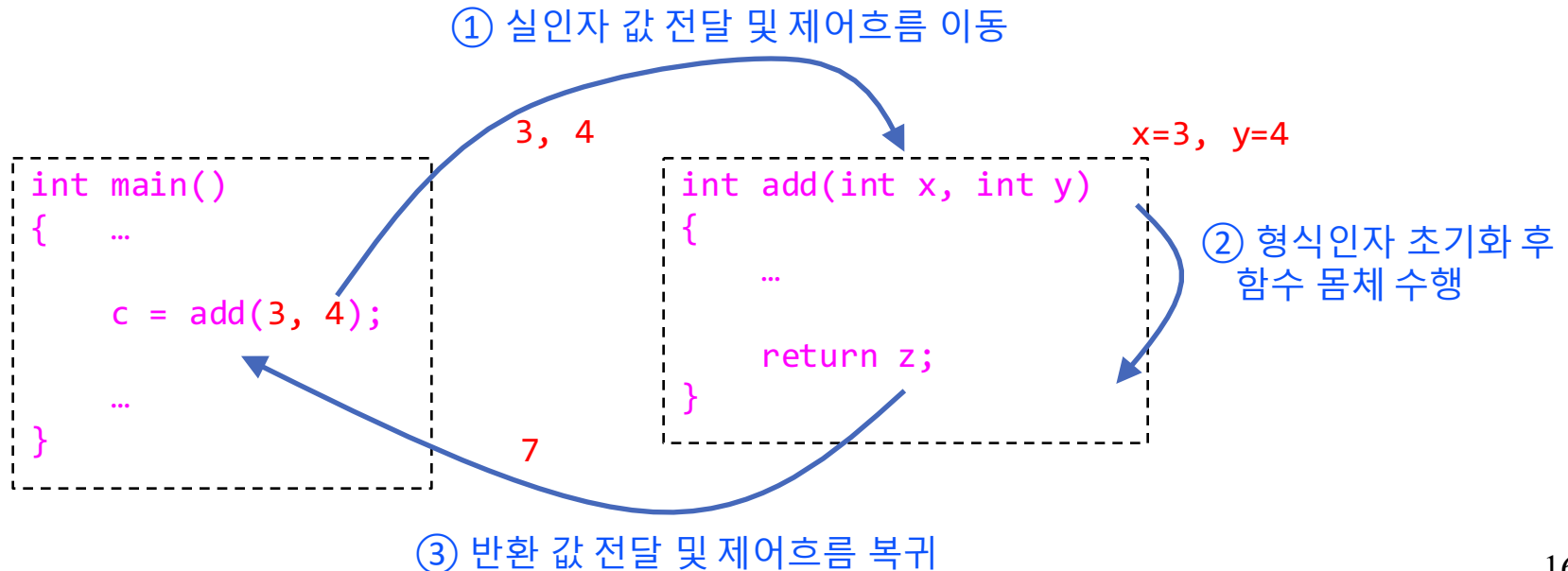
    return 0;
}
```

함수 이름      add    (3, 4)      인자에 전달할 값

### 3) 함수 호출과 반환

#### ■ 함수 호출 과정 분석

- ① main() 함수에서 add(3,4)를 호출 -> 인자 값 3과 4가 add 함수에 전달 되고 프로그램의 제어는 add()함수로 넘어감
- ② add() 함수에서는 인자를 x=3, y=4로 초기화(대입) 한 후, 함수의 몸체를 수행
  - 형식인자 : add()함수의 정의에 사용된 x, y
  - 실 인자 : add()함수 호출 시 넘겨 받는 값(3,4)
- ③ add() 함수가 종료되면, 프로그램 제어는 함수를 호출했던 라인(c=add(3,4))으로 복귀하고 나머지 부분 수행





### 3) 함수 호출과 반환

---

- **일반적인 함수 호출 과정(메커니즘) 요약**

- ① 실 인자 값 전달 및 제어흐름 이동

- 함수 A에서 함수 B를 호출하면, 호출 시 사용된 실 인자 값이 함수B에 전달되고 프로그램 제어가 함수 B로 넘어감

- ② 형식 인자 초기화 후 함수 몸체 수행

- 함수 B는 형식 인자를 실 인자의 값으로 초기화하고 몸체 수행
    - 수행 도중 return문을 만나거나 함수의 끝에 도달하면 함수 B는 종료
    - return문을 만나 종료 시 반환 값이 있을 경우 함수 A에 전달

- ③ 반환 값 전달 및 제어흐름 복귀

- 함수 B가 종료되면, 프로그램 제어는 함수 A로 복귀
    - 함수 B의 반환 값이 함수 B의 호출 결과로 사용됨

### 3) 함수 호출과 반환

- 함수 호출 과정 확인

- 다음과 같이 각 함수의 시작과 끝에 printf문을 삽입하여 함수 호출 시 제어흐름과 값이 전달되는 과정을 확인해보자.

```
int add(int x, int y)
{
    int z = -1;
    printf("add start: x=%d, y=%d, z=%d\n", x,y,z);

    z = x + y;

    printf("add end: x=%d, y=%d, z=%d\n", x,y,z);
    return z;
}
```

```
int main()
{
    int c=0;
    printf("main start: c=%d\n", c);

    c = add(3, 4); // 함수 호출부

    printf("main end: c=%d\n", c);
    return 0;
}
```

### 3) 함수 호출과 반환

- 함수 호출과 반환의 다양한 형태

- 실인자의 다양한 형태
- 이전 슬라이드와 마찬가지로, printf문을 이용하여 함수 호출 과정을 확인하라.

```
int add(int x, int y)
{
    ...
    return z;
}
```

```
int main()
{
    int a=4, b=3;
    int v1,v2,v3,v4,sum;

    v1 = add(a, a+b);
    v2 = add(1, a+2);
    v3 = add(1+2, a) - 3;
    sum = add(1, b)+add(a, 2);
    v4 = add(a, add(1, 2));

    return 0;
}
```

### 3) 함수 호출과 반환

- 함수 호출과 반환의 다양한 형태
  - main( ) → func( ) → add( )

```
int func(int a, int b)
{
    int z = add(a,b);

    if(z > 0) return 1;
    if(z < 0) return -1;
    return 0;
}
```

다양한  
형태의 반환

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

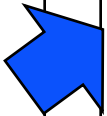
```
int main()
{
    int c;
    c = func(1,2);
    return 0;
}
```

### 3) 함수 호출과 반환


- **printf문을 이용한 함수 호출 과정 확인**
  - 이를 위해 초기화 등 코드 약간 수정

```
int func(int a, int b)
{
    int z = 0;
    printf(...);
    z = add(a,b);

    if(z > 0){
        printf(...);
        return 1;
    }
    if(z < 0){
        printf(...);
        return -1;
    }
    printf(...);
    return 0;
}
```



```
int add(int x, int y)
{
    int z=0;
    printf(...);
    z = x+y;
    printf(...);
    return z;
}
```



```
int main()
{
    int c=-1;
    printf(...);
    c = func(1,2);
    printf(...);
    return 0;
}
```

### 3) 함수 호출과 반환

---

- [실습6] 12~14쪽 슬라이드에 정의된 함수를 이용하여 다음 프로그램을 작성하시오.
  - 문자 'a' 는 한 번, 문자 'b' 는 두 번, ..., 문자 'z'는 26번 출력하시오.
    - ✓ 각 문자별 한 줄에 하나씩 출력
    - ✓ `print_characters` 함수를 반복 호출

출력

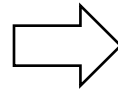
```
a
bb
ccc
dddd
... (생략)
```

### 3) 함수 호출과 반환

- [실습7] 12~14쪽 슬라이드에 정의된 함수를 이용하여 다음 프로그램을 작성하시오.
  - 4개의 정수 a, b, c, d를 입력 받아, 이 중 최댓값을 출력하시오.
    - ① a와 b 중 큰 값을 찾기 위해 **max** 함수 호출
    - ② c와 d 중 큰 값을 찾기 위해 **max** 함수 호출
    - ③ 위 두 결과값 중 큰 값을 찾기 위해 **max** 함수 호출

입력 예시1

5 2 9 4



출력 예시1

9

- ✓ (추가) max함수의 인자에서 max함수를 호출하는 방식으로, 하나의 문장(수식)만을 사용하여 최댓값을 찾도록 프로그램을 수정해 보시오.

### 3) 함수 호출과 반환

- 함수 원형 선언 (함수 선언)

- 왼쪽, 오른쪽의 코드를 각각 컴파일 했을 시 결과는?
  - 함수 정의 순서만 다르고 나머지는 동일

```
int add (int x, int y)
{
    . . .
}

int main()
{
    . . .
    c = add(3,4);
    . . .
}
```

컴파일 성공

```
int main()
{
    . . .
    c = add(3,4);
    . . .
}

int add (int x, int y)
{
    . . .
}
```

오류 발생  
add()가 정의  
되지 않았습니다.

컴파일 오류 or 경고



### 3) 함수 호출과 반환

#### ▪ 함수 원형 선언 (함수 선언)

```
int main()
{
    . . .
    c = add(3,4);
    . . .
}

int add (int x, int y)
{
    . . .
}
```

오류 발생  
add()가 정의  
되지 않았습니다.

- 코드를 컴파일 했을 시 컴파일 오류or 경고가 발생
- C 프로그램에서는 함수 정의를 함수 호출 위치보다 앞에 작성 해주어야 함
- 하지만, 다수의 함수를 정의해 사용할 때, 함수 호출 순서에 따라 함수를 배치하는 것은 프로그램 작성이 불편
- 이 문제를 해결하는 방법은?  
(다음 슬라이드)

### 3) 함수 호출과 반환

- 함수 원형 선언 (함수 선언)

```
int add (int x, int y); ← 함수 원형 선언

int main(){
    . . .
    c = add(3,4); ← 정상 동작
    . . .
}

int add (int x, int y){ ← 함수 정의
    . . .
}
```

- 함수의 형태를 표현하는 함수 원형을 코드의 앞 부분에 선언
- 원형 선언한 함수가 어딘가에 정의 되어 있다는 것을 알려줌
- 즉, 인자가 두 개의 int형 변수이고, 반환형이 int형인 add() 함수가 어딘가 정의 되어있다는 것을 알려주는 역할

### 3) 함수 호출과 반환

- **함수 원형 선언 (함수 선언)**

- 함수의 헤더와 동일한 형태를 가지는데, 마지막에 세미콜론(;)을 붙여줌

반환형 함수이름 ( 인자선언1, 인자선언2,...) ;

- 함수의 형태를 지정해 주는 것이므로, 인자 선언에서 변수명은 무시됨
- 인자 이름을 생략해도 되고, 심지어 함수 정의에 사용된 변수명과 다른 변수명 명시 가능

```
int add(int x, int y);  
int add(int a, int b);  
int add(int, int);
```

- 위 3개의 선언은 모두 동일

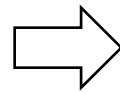
### 3) 함수 호출과 반환

---

- [실습8] 12~14쪽 슬라이드에 정의된 함수를 이용하여 다음 프로그램을 작성하시오. (함수 원형 선언 사용)
  - 영문자 10개를 입력 받아, 모두 대문자로 변환하여 출력하시오.
    - ✓ 소문자를 대문자로 변환하기 위해 `atoA` 함수를 반복 호출

입력 예시1

He1l0WorLd



출력 예시1

HELLOWORLD

### 3) 함수 호출과 반환

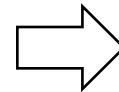
- [실습9] 12~14쪽 슬라이드에 정의된 함수를 이용하여 다음 프로그램을 작성하시오. (함수 원형 선언 사용)
  - 6개의 정수 a, b, c, d, e, f를 입력 받아, 다음 수식의 결과를 출력하시오.

$$a/b + c/d + e/f \quad (\text{실수 연산 수행})$$

- ✓ 분수를 계산하기 위해 **divide** 함수 호출
- ✓ 덧셈을 계산하기 위해 **add3** 함수 호출

입력 예시1

5 2 9 4 6 4



출력 예시1

6.25

- ✓ (추가) add3함수의 인자에서 divide함수를 호출하는 방식으로, 하나의 문장(수식)만을 사용하여 위 수식의 결과를 계산하도록 프로그램을 수정해 보시오.

## 4) 지역 변수와 전역 변수

---

- **변수의 적용범위**

- 함수 뿐만 아니라 함수에서 사용하는 변수도 **독립성**이 적용
- 즉, 함수에서 선언된 변수들은 그 함수에서만 유효
- 경우에 따라서 특정 함수에만 국한되지 않고, 함수와 무관하게 사용되는 변수가 필요

- **변수 종류**

- 지역변수
- 전역변수

## 4) 지역 변수와 전역 변수

### ■ 지역 변수

- ✓ 함수 내에서 선언한 변수
- ✓ 변수를 선언한 함수 내에서만(지역적으로) 유효
- ✓ 함수 호출과 동시에 자동으로 생성되고 함수가 종료되면 자동으로 소멸되어 **자동변수**라고도 함
- ✓ add()함수의 인자로 사용된 변수 x와 y도 add()함수의 지역변수

- 다음 코드는 왜 컴파일 오류가 발생 할까?

컴파일 오류

```
int add (int x, int y)
{
    int c;
    → c = x + y;
    return c;
}
```

```
int main()
{
    int c;
    c = add(3,4);
    . . .
}
```

## 4) 지역 변수와 전역 변수

- 다음 코드의 결과는? 왜 이런 결과가 나오는지 생각해 보자

```
int add (int x, int y)
{
    int c;

    c = x + y;

    return c;
}
```

```
int main()
{
    int c = 10;

    printf("3 + 4 = %d\n", add(3,4));
    printf("c = %d\n", c);
    return 0;
}
```

[실행결과]

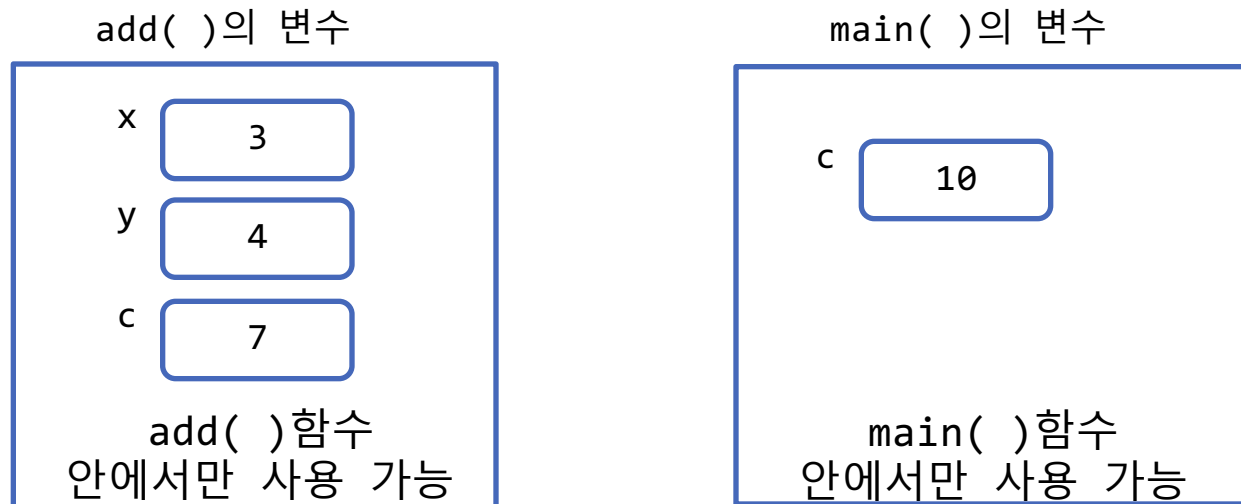
3 + 4 = 7  
c = 10

- 지역 변수의 성질에 유의하여 각 함수에서 c의 값을 생각해 보자



## 4) 지역 변수와 전역 변수

- 설명
  - ✓ 총 4개의 지역 변수가 존재
  - ✓ main() 함수에서 선언된 변수 c와 add()함수에서 선언된 변수 c는 이름은 같지만 서로 다른 변수
  - ✓ add함수에서 c의 값의 변경이 main함수의 c에 영향을 끼치지 않음



- 지역 변수의 특성은 함수의 정의 구현에 독립성을 부여해줌
  - ✓ 다른 함수에서 어떤 변수 이름이 사용됐는지 고려할 필요 없음

## 4) 지역 변수와 전역 변수

### ■ 전역 변수

- ✓ 지역 변수 이외에 프로그램 내 어디서든 사용할 수 있는 변수
- ✓ 전역 변수는 함수 밖에서 선언
- ✓ **자동으로 0으로 초기화**
  - ✓ 자동으로 초기화해 주지만, 모든 변수는 명시적으로 초기화하는 습관을 가지자.

**int c;**      ← 전역변수

```
int add (int x, int y)
{
    c = x + y;

    return c;
}
```

## 4) 지역 변수와 전역 변수

- 다음 코드의 결과는? 왜 이런 결과가 나오는지 생각해 보자

**int c;**      ← 전역변수

```
int add (int x, int y)
{
    c = x + y;

    return c;
}
```

```
int main()
{
    c = 10;

    printf("3 + 4 = %d\n", add(3,4));
    printf("c = %d\n", c);

    return 0;
}
```

[실행결과]

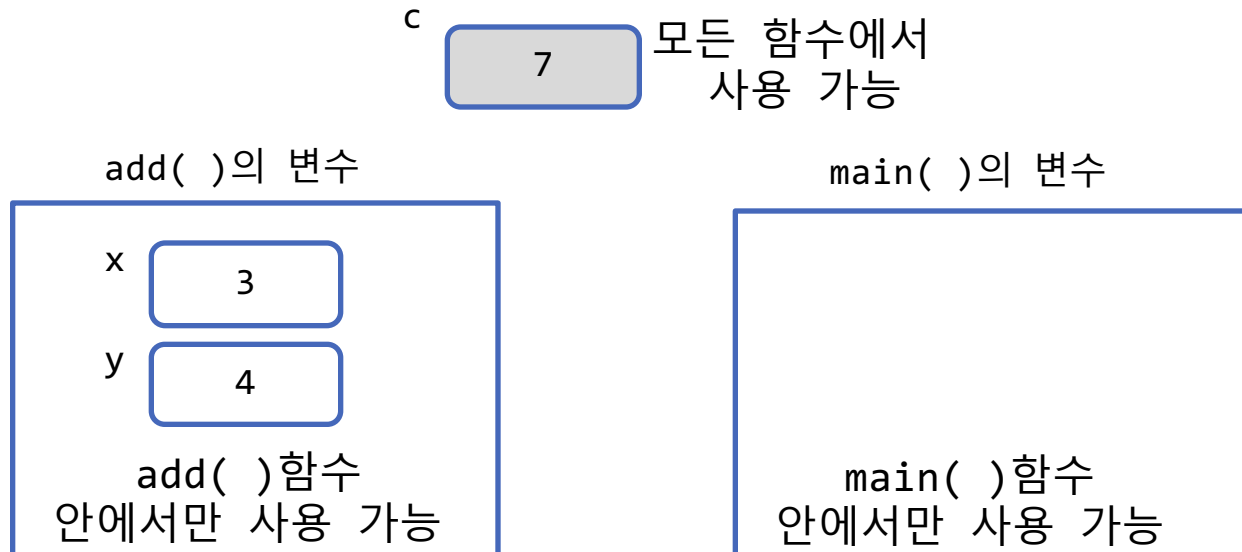
3 + 4 = 7  
c = 7

- 전역 변수의 성질을 유의하여 동작과정을 생각해 보자

## 4) 지역 변수와 전역 변수

- 설명

- ✓ 총 2개의 지역 변수, 1개의 전역변수 존재
- ✓ 변수 c는 함수 밖에서 선언 된 전역 변수
- ✓ 변수 c는 프로그램 전체에서 유효하기 때문에 모든 함수에서 사용가능
- ✓ 따라서 add에서 사용한 c는 main에서 사용한 c와 동일하므로 같은 값이 출력



## 4) 지역 변수와 전역 변수

- 코드를 작성해보고 컴파일 오류가 생기는지 확인해보자

```
int c;    ← 전역변수
```

```
void add (int x, int y)
{
    c = x + y;

    printf("add: c = %d\n", c);
}
```

```
int main()
{
    int c = 10;

    add(3,4);
    printf("main: c = %d\n", c);

    return 0;
}
```

[실행결과]

```
add: c = 7
main: c = 10
```

- 전역 변수와 지역변수의 이름이 동일해도 컴파일 성공
- 실행결과는 왜 이렇게 나왔을까?

## 4) 지역 변수와 전역 변수

- 설명

- ✓ 지역 변수와 전역 변수가 동일한 이름으로 선언 되면, 지역 변수가 우선시 됨
- ✓ 즉, add()함수에서 사용한 변수 c는 전역변수, main() 함수에서 사용한 변수 c는 지역변수
- ✓ add(3,4)함수에서 전역 변수 c에 3+4 값을 대입 하더라도 main() 함수의 지역변수 c에 어떠한 영향을 주지 못함

c 7 모든 함수에서  
사용 가능

add( )의 변수

x 3  
y 4

add( )함수  
안에서만 사용 가능

main( )의 변수

c 10

main( )함수  
안에서만 사용 가능

## 4) 지역 변수와 전역 변수

### ■ 전역 변수의 역할

- 아래 코드와 같이 add()함수의 결과 값을 main() 함수에 전달하기 위해, return 문 대신 전역 변수를 이용 가능

```
int c;    ← 전역변수

void add (int x, int y)
{
    c = x + y;
}
```

```
int main()
{
    add(3,4)
    printf("3 + 4 = %d\n", c);

    return 0;
}
```

[실행결과]

3 + 4 = 7

## 4) 지역 변수와 전역 변수

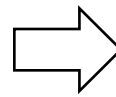
- [실습10] 다음과 같이 함수를 정의하고 사용하시오.
  - div( ) 함수
    - ✓ 반환형은 int, 인자는 int형 변수 2개
    - ✓ 인자 2개를 나눈 몫을 반환, 나머지는 전역변수에 저장
  - main( ) 함수
    - ✓ 두 개의 정수를 입력 받고, div 함수를 호출하여 몫과 나머지 계산하여 한 줄에 출력

입력 예시1

5 3

입력 예시2

4 2



출력 예시1

1 2

출력 예시2

2 0



## 4) 지역 변수와 전역 변수

---

- [실습11] 다음과 같이 함수를 정의하고 사용하시오.
  - swap( ) 함수
    - ✓ 반환형은 void형, 인자는 없음
    - ✓ **전역변수 a**와 **b**의 값을 교환 (즉, 두 개의 변수 값 바꾸기)
  - main() 함수
    - ✓ 두 개의 정수를 입력 받아 전역 변수 a와 b에 저장
    - ✓ swap( ) 함수를 호출
    - ✓ 교환된 두 전역 변수의 값 출력

실행 예  
(붉은 색은 사용자 입력)

```
a 입력: 6
b 입력: 8
swap 함수 호출 후
a = 8
b = 6
```

## 4) 지역 변수와 전역 변수

### ■ 전역변수 사용시 주의점

- 전역 변수는 편리하나, 함수의 독립성을 해침
- 큰 프로그램을 작성하는 경우, 전역 변수 사용은 신중히
- 함수 하나를 작성하기 위해서는 호출하는 함수뿐만 아니라 다른 모든 함수에서 전역 변수가 어떻게 사용되는 지 고려해야 함

### ■ 지역 변수와 전역변수 비교

	지역변수	전역변수
선언 위치	함수 내부	함수 외부(밖)
유효 범위	선언한 함수 내부에서만 사용 가능	프로그램 내 어디서든지 사용 가능
자동 초기화	X (사용자가 직접 초기화)	O (0으로 자동 초기화)
변수 소멸 시점	해당 함수가 종료될 때	프로그램이 종료될 때

## 5) 배열 인자

- 배열의 원소를 일반 변수처럼 함수에 전달한다.

```
#include <stdio.h>
void max(int a, int b){
    if (a > b) printf("%d ", a);
    else printf("%d ", b);
    printf("\n");
}

int main(void){
    int score[3]={70, 80, 90};
    max(score[0], score[1]);
    return 0;
}
```

80

- 주의: 배열의 인덱스 범위 (여기서는 0~2) 밖에 있는 배열 원소를 사용하거나, 함수에 전달하지 않도록 한다.
  - ✓ 쓰면 안 되는 예) max(score[3], score[4]) ;

## 5) 배열 인자

- 함수에 **배열 전체**를 전달하는 경우, 함수 **실인자**로서 배열이름만 쓴다.

```
#include <stdio.h>
void print(int ar[3]){
    int i;
    for(i=0;i<3;i++) printf(" %d", ar[i]);
    printf("\n");
}

int main(void){
    int score[3]={70, 80, 90};
    print(score);
    return 0;
}
```

70 80 90

## 5) 배열 인자

---

- 함수 형식인자로서 배열 크기는 생략해도 된다.

```
#include <stdio.h>
void print(int ar[ ]){
    int i;
    for(i=0;i<3;i++) printf("%d ", ar[i]);
    printf("\n");
}

int main(void){
    int score[3]={70, 80, 90};
    print(score);
    return 0;
}
```

## 5) 배열 인자

---

- 함수 형식인자로서 배열 크기는 생략해도 된다.

```
#include <stdio.h>
void print(int ar[ ]){
    int i;
    for(i=0;i<3;i++) printf("%d ", ar[i]);
    printf("\n");
}

int main(void){
    int score[3]={70, 80, 90};
    print(score);
    return 0;
}
```

## 5) 배열 인자

- 배열 **크기**는 함수의 인자로 별도로 넘겨주어야 한다.

```
#include <stdio.h>
void print(int ar[ ], int size){
    int i;
    for(i=0; i<size; i++) printf("%d ", ar[i]);
    printf("\n");
}

int main(void){
    int score[3]={70, 80, 90};
    print(score, 3);
    return 0;
}
```

## 5) 배열 인자

- 배열 크기가 변경되는 경우를 대비하여 배열 **크기**를 자동으로 계산하여 함수에 전달할 수 있다.

```
#include <stdio.h>
void print(int ar[ ], int size){
    int i;
    for(i=0;i<size;i++) printf("%d ", ar[i]);
    printf("\n");
}

int main(void){
    int score[3]={70, 80, 90};
    print(score, sizeof(score)/sizeof(int));
    return 0;
}
```



## 6) 함수와 라이브러리

---

- **라이브러리**
  - 함수들을 구현해 모아 놓은 것
  - 필요 시 함수를 호출하여 사용
- **표준 라이브러리**
  - C언어에서 정해놓은 표준 함수들로 구성: printf(), scanf()
- **표준함수 사용**
  - 함수의 형태와 기능만 알고 있으면 활용할 수 있음
  - 어떻게 구현되어 있는지는 몰라도 됨
  - 다만, 호출하기 전에 함수 원형이 선언되어 있어야 함
  - printf(), scanf() 함수의 원형 선언은 어디에 있을까?
    - ✓ 다음 장에서 계속

## 6) 함수와 라이브러리

---

- 다음 코드를 컴파일하면?

```
#include <stdio.h>    // 이 부분 삭제

int main()
{
    printf("Hello, World!!\n");
    return 0;
}
```

✓ printf() 함수를 찾을 수 없다고 컴파일 오류 발생

- #include 문은 stdio.h 파일을 소스코드에 포함시키라는 의미
- printf() 함수의 원형은 stdio.h 파일에 선언되어 있음
- stdio.h 를 헤더파일이라 부름 ( 확장자 .h)

## 6) 함수와 라이브러리

- **표준 함수와 헤더파일**

- ✓ 표준 함수를 사용하기 위해서는 적절한 헤더 파일을 `#include` 문을 이용해 소스 코드에 포함시켜야 함

- **자주 사용되는 C 표준 헤더 파일 및 표준 함수**

- ✓ 일부 함수는 뒷장에서 학습
- ✓ 자세한 내용은 개발 툴의 도움말이나 C 표준 문서 참고

헤더파일	포함된 함수의 기능	표준 함수
stdio.h	입력, 출력, 파일	printf, scanf, putc, getc, fopen 등
stdlib.h	숫자변환, 동적 할당	atoi, rand, srand, malloc, free 등
ctype.h	문자 검사 및 변환	isalnum, isalpha, islower, toupper 등
math.h	수학 함수	sin, asin, exp, log, pow, sqrt, abs 등
time.h	시간 처리	clock, time, difftime 등
string.h	문자열, 메모리 블록	strcpy, strcat, strcmp, strlen, memcpy 등

- 
- GCC에서는 `abs`가 `int` 입력 `int` 출력 함수로 인식됨  
VS 컴파일러에서는 `abs`가 `float` 입력 `float` 출력 함수로 인식됨  
이로 인한 출력 값 차이 발생 (OJ는 GCC를 사용함)

	함수 선언
VS	<code>float abs(float a);</code>
OJ, GCC	<code>int abs(int a);</code>