
C 프로그래밍 및 실습

8. 함수

세종대학교

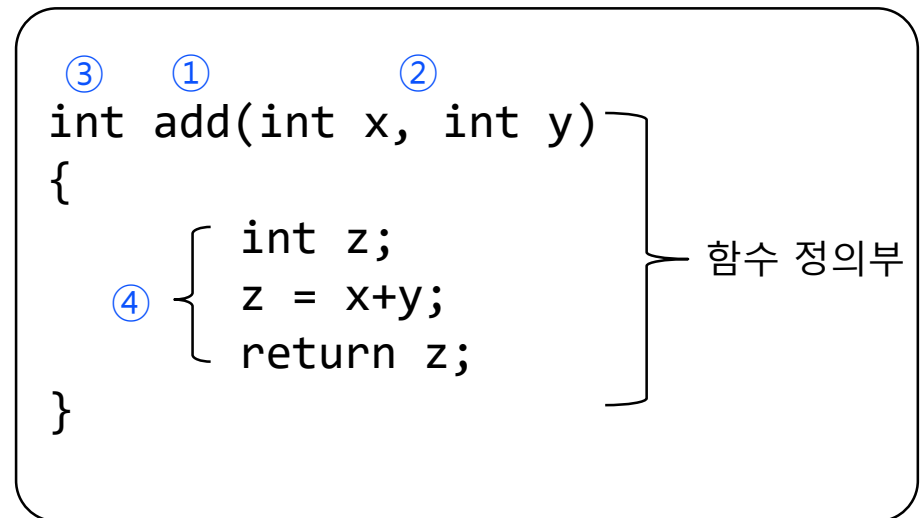
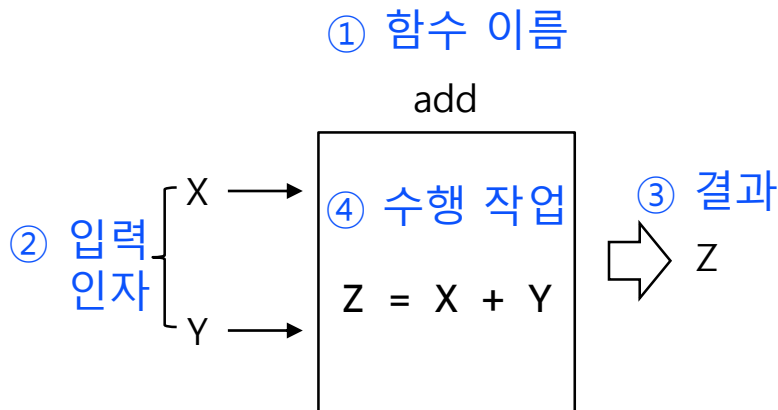
목차

- 1) 함수 개요
- 2) 함수 정의
- 3) 함수 호출과 반환
- 4) 함수와 변수의 종류
- 5) 함수에 배열 전달
- 6) 함수와 라이브러리

1) 함수 개요

■ 수학의 함수와 C언어의 함수 개략적 비교

- $\text{add}(x,y) = x + y$: 두 수의 합을 구하는 함수
 - ✓ 함수(마술상자) `add`에 `x` 와 `y`를 입력하면 두수의 합이 결과로 나옴



- 구성요소 4가지가 전부 포함됨을 짐작할 수 있음

1) 함수 개요

- C언어에서 함수

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

- 함수는 함수 정의부와 함수 호출부로 구성

- ✓ 함수 정의부 : 함수를 구현하는 부분 (즉, 마술상자 내부 구현)

- ✓ 예) main() 함수 구현

- ✓ 함수 호출부 : 함수의 기능을 이용하는 부분 (즉, 마술상자 이용)

- ✓ 예) printf() 함수 호출

2) 함수 정의

- 함수 정의

- 함수가 수행해야 할 기능 명세하기

- 구문

```
반환형    함수이름    (인자선언1, 인자선언2, ...)  
{  
    함수의 수행코드  
  
    return 문;        ⇒ 필요 시  
}
```

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

2) 함수 정의

- 함수 정의 요소

- 함수 이름

- ✓ 함수 이름 규칙은 변수명 규칙과 동일 (식별자)
 - ✓ 의미있는 이름 사용 권장

- 함수 인자 or 매개 변수(parameter)

- ✓ 함수의 입력 데이터를 나타내는 변수들
 - ✓ 인자가 다수인 경우 콤마(,)로 구분
 - ✓ 인자가 없더라도 소괄호는 반드시 필요

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

    return z;
}
```

- 반환형(return type)

- ✓ 함수는 수행된 후에 반환하는 결과의 자료형
 - ✓ 아무 결과도 반환하지 않으면 자료형에 void를 씀

- 참고) 위 세 구성 요소는 함수의 형태 명세 → 함수 헤더라 칭함

2) 함수 정의

- 함수 정의 요소

- 함수 몸체(body)

- ✓ 함수가 수행해야 할 작업을 { } 안에 명세
 - ✓ 함수는 제어 흐름에 따라 수행되다, 맨 마지막 문장까지 수행 되거나 **return** 문을 만나면 종료

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

    return z;
}
```

- ✓ **return** 문

- ✓ 함수를 종료하고 함수의 결과를 반환하는 역할
 - ✓ 반환형이 void 인 경우 보통 return 문 사용하지 않지만, 중간에 함수를 종료시키기 위해 쓰기도 함
 - » return z; ⇨ 변수 z에 저장된 값 반환
 - » return 10+20; ⇨ 10+20, 즉 30을 반환
 - » return; ⇨ 아무 것도 반환하지 않음
(반환형이 void 인 경우 사용)

2) 함수 정의

▪ 함수 정의 예시 1

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

    . . .

    return c1;
}
```

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

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

▪ 함수 정의 예시 2

```
void print_heading( void )
{

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

}
```

1. 반환형 : 없음(void)
2. 함수 이름 : print_heading
3. 함수 인자 : 없음(void)

※ 반환형이 void형이므로 return문 없어도 됨

※ 인자가 없는 경우, void를 명시적으로 적어도 되고, 생략해도 됨

2) 함수 정의

- [예제 8.1] 다음 조건을 만족하는 함수를 정의 하시오.
 1. max() 함수
 - ✓ 함수 이름 : max
 - ✓ 인자 : int형 변수 a와 b
 - ✓ 반환형 : int형
 - ✓ a와 b 중 큰 값을 반환
 2. 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

3. 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)

4. add3 함수

- ✓ 함수 이름 : add3
- ✓ 인자 : float형 변수 a, b, c
- ✓ 반환형 : float
- ✓ a, b, c의 합을 반환

2) 함수 정의

5. atoi 함수

- ✓ 함수 이름 : atoi
- ✓ 인자 : char형 변수 ch
- ✓ 반환형 : int
- ✓ 소문자인 ch를 대문자로 변환하여 반환

→ 여기서 정의한 함수는 뒤의 예제에서 활용됨

목차

- 1) 함수 개요
- 2) 함수 정의
- 3) **함수 호출과 반환**
- 4) 함수와 변수의 종류
- 5) 함수에 배열 전달
- 6) 함수와 라이브러리

3) 함수 호출과 반환

- 함수 호출(사용) 방법

- 함수 이름을 쓰고, 소괄호 안에 함수 인자에 넣을 값을 차례로 적음

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



    z = x+y;

    return z;
}
```

```
int main()
{
    int c;

    c = add(3, 4); // 함수 호출부

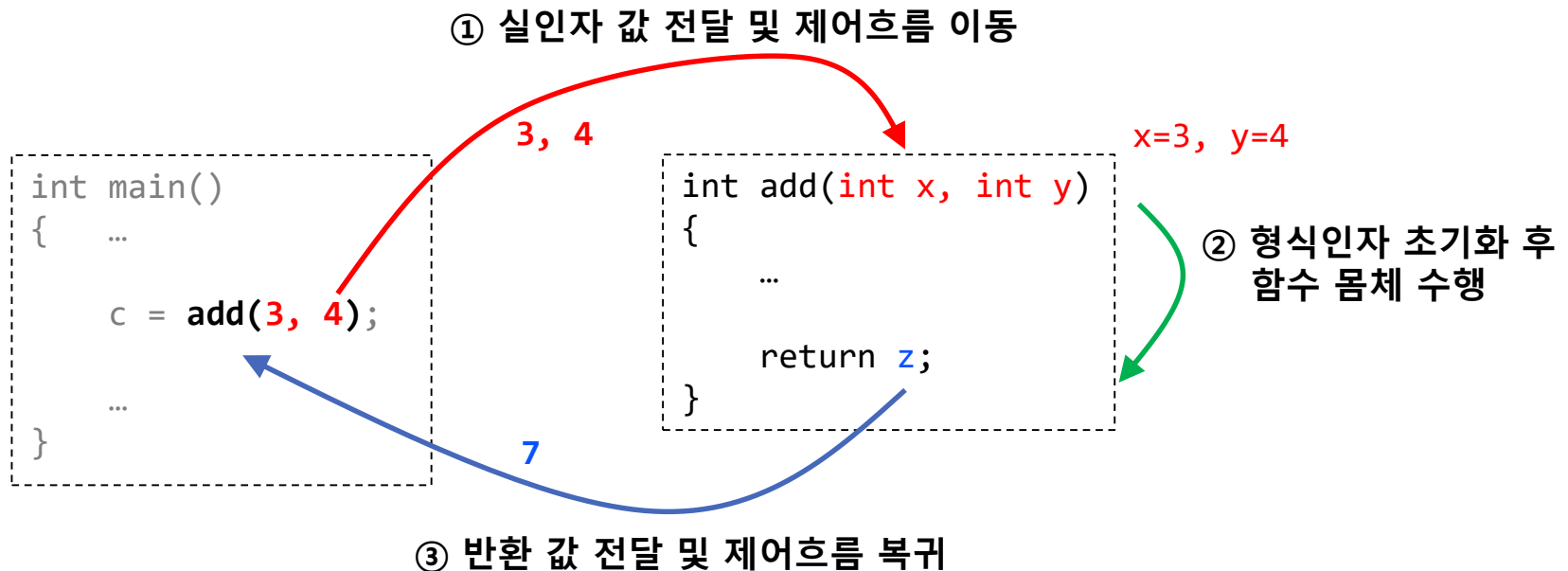
    return 0;
}
```

함수 이름  add (3, 4)  인자로 전달할 값

3) 함수 호출과 반환

■ 함수 호출 과정

- ① **add() 함수 호출:** 인자 값 3과 4가 add 함수에 전달되고, 프로그램의 제어는 add()함수로 넘어감
- ② **add() 함수 수행:** 인자를 x=3, y=4로 초기화(대입) 한 후, 몸체 수행
 - ✓ 형식인자 : add()함수의 정의에 사용된 x, y
 - ✓ 실 인자 : add()함수 호출 시 넘겨 받는 값(3,4)
- ③ **add() 함수 종료:** 프로그램 제어는 함수를 호출했던 라인(c=add(3,4))으로 복귀하고, 반환값을 사용해 나머지 부분 수행



3) 함수 호출과 반환

- 함수 호출 과정 확인

- 다음과 같이 각 함수의 시작과 끝에 printf문을 삽입하여 함수 호출 시 제어흐름과 값이 전달되는 과정을 확인해보자.
 - ✓ 함수 인자 전달과 반환이 제대로 수행되고 있는 지 점검하는 기본적인 방법

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

    z = x + y;

    printf("add end: z=%d\n", 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 호출의 다양한 형태

- 실인자의 다양한 형태
- printf문을 이용하여 함수 호출 과정을 확인해보자

```
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 호출의 다양한 형태
 - main() → func() → add()

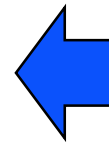
```
int add(int x, int y)
{
    return x+y;
}
```



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

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

} return 문
여러 개
사용됨



```
int main() {
    int c;

    c = func(1,2) ;
    printf("c = %d\n",c);

    return 0;
}
```

3) 함수 호출과 반환

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

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

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

3) 함수 호출과 반환

- [예제 8.2] (1~2번) 예제 8.1에서 정의된 함수를 이용하여 다음 프로그램을 작성하시오.
 1. 문자 'a' 는 한 번, 문자 'b' 는 두 번, ..., 문자 'z'는 26번 출력
 - ✓ 각 문자별 한 줄에 하나씩 출력
 - ✓ `print_characters` 함수를 반복 호출

실행 결과

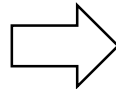
```
a
bb
ccc
dddd
... (생략)
```

3) 함수 호출과 반환

2. 4개의 정수 a, b, c, d를 입력 받아, 최댓값 출력
- 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)

- add() 함수 정의를 main() 함수 뒤에 작성하면?

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

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

오류 발생
add()가 정의
되지 않았습니다.

- 컴파일 오류 or 경고 발생
- 함수 정의를 함수 호출 위치보다 앞에 작성 해주어야 함
- 다수의 함수를 정의해 사용할 때, 함수 호출 순서에 맞추어 함수를 배치하는 것은 불편

3) 함수 호출과 반환

- 함수 원형 선언 (함수 선언)
 - 이 문제를 해결하는 방법은?

```
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 원형**을 앞 부분에 **선언**
- 인자가 두 개의 int형 변수이고, 반환형이 int형인 add() 함수가 어딘가 정의되어있다는 것을 알려주는 (선언) 역할
- 용어 '**선언**' 과 '**정의**'를 혼동하지 말자

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 명시하는 것도 가능
 - ✓ 아래 세 선언은 모두 명일

```
int add(int x, int y);  
int add(int a, int b);  
int add(int, int);    → 권장 형태
```

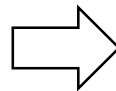
3) 함수 호출과 반환

- [예제 8.2] (3~4번) 예제 8.1에 정의된 함수를 이용하여 다음 프로그램을 작성하시오. (함수 원형 선언 사용)

3. 영문자 10개를 입력 받아, 모두 대문자로 변환하여 출력
✓ 소문자를 대문자로 변환하기 위해 **atoA** 함수를 반복 호출

입력 예시1

He1l0WorLd



출력 예시1

HELLOWORLD

3) 함수 호출과 반환

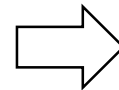
4. 6개의 정수 a, b, c, d, e, f를 입력 받아, 다음 수식의 결과를 출력하시오.

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

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

입력 예시1

5 2 9 4 6 4



출력 예시1

6.250000

- ✓ **(추가)** add3() 함수의 인자에서 divide() 함수를 호출하는 방식으로, 하나의 문장(수식)으로 수식의 결과 값이 계산되도록 프로그램을 수정해 보시오.

목차

- 1) 함수 개요
- 2) 함수 정의
- 3) 함수 호출과 반환
- 4) **함수와 변수의 종류**
- 5) 함수에 배열 전달
- 6) 함수와 라이브러리

4) 함수와 변수의 종류

- **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
- 정적변수

4) 함수와 변수의 종류

■ 지역 변수

- ✓ 선언 위치: 함수 내에서 선언
- ✓ 유효 범위: 변수를 선언한 함수 내에서만(지역적으로) 유효
- ✓ 함수 호출과 동시에 자동으로 생성되고 함수가 종료되면 자동으로 소멸되어 자동변수라고도 함
- ✓ 함수의 형식 인자도 지역변수임

- 다음 코드에서 컴파일 오류가 발생하는 이유는?

컴파일 오류 →

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

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

4) 함수와 변수의 종류

- 지역 변수의 독립성: 함수 구현에 독립성을 부여
 - 아래에서 두 함수의 변수 `c`는 서로 다른 변수

```
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;  
}
```

add()의 변수

x	3
y	4
c	7

add() 함수
안에서만 사용 가능

main()의 변수

c	10
---	----

main() 함수
안에서만 사용 가능

실행 결과

3	+	4	=	7
c	=	10		

4) 함수와 변수의 종류

■ 전역 변수

- ✓ 선언 위치: 함수 밖에서 선언
- ✓ 유효 범위: 프로그램 내 어디서든 사용 가능
- ✓ 자동으로 0으로 초기화
 - ✓ But, 모든 변수는 명시적으로 초기화 하는 습관을 가지자.

```
int c = 0; // 전역변수 선언
```

```
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;
}
```

실행 결과

```
3 + 4 = 7
c = 7
```

4) 함수와 변수의 종류

- 동일한 이름의 전역변수와 지역변수
 - 지역 변수가 우선

```
int c = 0; // 전역변수
```

```
void add (int x, int y) {  
    c = x + y;  
    printf("add: c = %d\n", c);  
}
```

```
int main() {  
    int c = 10; // 지역 변수  
    add(3,4);  
    printf("main: c = %d\n", c);  
    return 0;  
}
```

실행 결과

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

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) 함수와 변수의 종류

- 전역 변수는 함수 사이의 데이터 전달을 위한 또 하나의 수단
 - ✓ 아래 코드는 설명을 위한 예제로 좋은 방식의 코드는 아님

```
int c = 0; // 전역변수 선언

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

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

실행 결과

3 + 4 = 7

- 전역 변수를 사용하면 함수 사이의 데이터 전달이 편리하나,
함수의 독립성을 해치므로 신중을 기해야 함

4) 함수와 변수의 종류

- [예제 8.3] 다음과 같이 함수를 정의하고 사용하시오.

1. 몫과 나머지 계산 프로그램

- ✓ div() 함수

- ✓ 반환형은 int, 인자는 int형 변수 2개

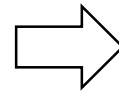
- ✓ 인자 2개를 나눈 몫을 반환, 나머지는 전역변수에 저장

- ✓ main() 함수

- ✓ 두 개의 정수를 입력 받고, div 함수를 호출하여 몫과 나머지 계산하여 한 줄에 출력

입력 예시1

5 3

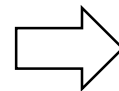


출력 예시1

1 2

입력 예시2

4 2



출력 예시2

2 0

4) 함수와 변수의 종류

2. 값 교환 프로그램

- ✓ swap() 함수

- ✓ 반환형은 void형, 인자는 없음

- ✓ 전역변수 a와 b의 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 예시
(붉은 색은 사용자 입력)

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

4) 함수와 변수의 종류

■ 변수의 지속시간

- ✓ 지역 변수: 함수 호출 시 생성, 함수 종료 시 소멸
- ✓ 전역 변수: 프로그램 실행 시 생성, 프로그램 종료 시 소멸

```
void inc_L() {  
    int c = 0;    ⇒ 지역 변수  
    ++c;  
    printf("%d\n", c);  
}  
int main(){  
    inc_L();    ⇒ 첫 번째 호출  
    inc_L();    ⇒ 두 번째 호출  
    inc_L();    ⇒ 세 번째 호출  
    return 0;  
}
```

실행 결과

```
1 ⇒ 첫 번째 호출  
1 ⇒ 두 번째 호출  
1 ⇒ 세 번째 호출
```

```
int c = 0;    ⇒ 전역 변수  
void inc_G() {  
    ++c;  
    printf("%d\n", c);  
}  
int main(){  
    inc_G();    ⇒ 첫 번째 호출  
    inc_G();    ⇒ 두 번째 호출  
    inc_G();    ⇒ 세 번째 호출  
    return 0;  
}
```

실행 결과

```
1 ⇒ 첫 번째 호출  
2 ⇒ 두 번째 호출  
3 ⇒ 세 번째 호출
```

4) 함수와 변수의 종류

■ 정적 변수

- **static** 키워드 사용

- ✓ 선언 위치와 유효 범위: 지역 변수와 동일
 - ✓ 함수내 선언, 선언한 함수 내부에서만 사용 가능
- ✓ 지속 시간: 전역 변수와 동일
 - ✓ 프로그램 실행 전체 과정 동안 딱 한번만 생성되고 초기화

```
void inc_S() {  
    static int c = 0;    ⇒ 정적 변수  
  
    ++c;  
    printf("%d\n", c);  
}  
int main(){  
    inc_S();    ⇒ 첫 번째 호출  
    inc_S();    ⇒ 두 번째 호출  
    inc_S();    ⇒ 세 번째 호출  
}
```

실행 결과

1	⇒	첫 번째 호출
2	⇒	두 번째 호출
3	⇒	세 번째 호출

4) 함수와 변수의 종류

- 함수와 관련된 변수 종류와 특징

	지역 변수	정적 (지역) 변수	전역 변수
선언 위치	함수 내부		함수 외부
사용 범위	선언한 함수 내부에서만 사용 가능		프로그램 내 어디서든지 사용가능
자동 초기화	x (사용자가 직접 초기화)	o (0으로 자동 초기화)	
지속 시간	함수 호출될 때마다 생성, 해당 함수 종료 시 소멸	프로그램이 실행 동안 단 한번 생성, 프로그램 종료 시 소멸	

목차

- 1) 함수 개요
- 2) 함수 정의
- 3) 함수 호출과 반환
- 4) 함수와 변수의 종류
- 5) **함수에 배열 전달**
- 6) 함수와 라이브러리

5) 함수에 배열 전달

배열의 개별 원소 전달

- 함수의 인자에서 사용된 배열 원소는 수식의 일부일 뿐
✓ 배열 원소는 일반 변수와 동일하게 취급됨(7장)

```
void print_int( int x ) {           ⇒ 전달되는 값이 정수형이므로 int
    printf("전달된 값 : %d\n", x);
}
int main() {
    int a[3] = { 10, 20, 30 } ;

    print_int( a[1] );               ⇒ a[1]에 저장된 값이 전달됨
    print_int( a[0] + a[2] + 100 ); ⇒ 연산의 결과가 전달됨

    return 0;
}
```

실행 결과

전달된 값 : 20
전달된 값 : 140

5) 함수에 배열 전달

- 배열 전체 전달 (일차원 배열)

✓ 형식인자에서 배열 크기 명시해도 의미 없음 → 보통 생략

```
void print_arr( int x[ ] ) {           ⇒ 인자의 자료형은 배열로  
    int i;                             (크기는 생략)  
    printf("전달된 배열 :");  
    for( i=0; i < 3 ; ++i )  
        printf(" %d", x[i]);          ⇒ 배열의 원소 값 출력  
}  
int main() {  
    int a[3] = { 10, 20, 30 } ;  
    print_arr( a );                    ⇒ 인자로 배열 전달 (이름만 씀)  
    return 0;  
}
```

실행 결과

전달된 배열 : 10 20 30

5) 함수에 배열 전달

- 배열 크기가 필요한 경우
 - 별도의 함수 인자로 전달

```
void print_arr( int x[ ], int n ) { ⇒ 배열과 배열의 크기 전달
    int i;
    printf("전달된 배열 :");
    for( i=0; i < n ; ++i )
        printf(" %d", x[i]);           ⇒ 배열의 원소 값 출력
    printf("\n");
}
int main() {
    int a[3] = { 10, 20, 30 };
    int b[5] = { 11, 22, 33, 44, 55 };

    print_arr( a , 3 );   ⇒ 배열 a와 a의 크기 전달 (동일한 함수 호출)
    print_arr( b , 5 );   ⇒ 배열 b와 b의 크기 전달 (동일한 함수 호출)

    print_arr( a, sizeof(a)/sizeof(int) ); ⇒ sizeof 연산자를 이용하여
                                           배열 크기 자동 계산

    return 0;
}
```

5) 함수에 배열 전달

- 배열 전체가 전달된 경우, 호출 함수에서 배열의 값을 바꾸면?
 - 원 함수의 배열 값도 바뀜

```
void print_arr( int x[ ], int n ) { ... } ⇒ 배열 출력 함수 (생략)

void zero_arr_elem( int x[ ], int i ) { ⇒ 배열이 전달됨
    x[i] = 0;                               ⇒ x[i]의 원소 값을 0으로
}

int main() {
    int a[3] = { 11, 22, 33 };

    zero_arr_elem( a , 1 );                ⇒ 배열 전달
    print_arr( a, sizeof(a)/sizeof(int) );

    return 0;
}
```

실행 결과

전달된 배열 : 11 0 33

5) 함수에 배열 전달

- 배열 자체를 인자로 전달하기(정리)
 - 다차원 배열도 유사 (자세한 내용은 교재 참고)
 - 배열 자체를 인자로 전달할 때는 일반 변수와 몇 가지 다른 점 존재
 - ✓ 형식 인자 형태
 - ✓ 배열 값 변경의 종속성
 - 실제 함수 호출 과정은 일반 변수와 동일하나, 표면적으로 달라 보이는 것일 뿐
 - 예외적으로 보이는 이유는 배열 자체를 전달한다는 것은 배열의 주소를 전달하기 때문
 - ✓ 이에 대해서는 포인터 단원에서 학습

목차

- 1) 함수 개요
- 2) 함수 정의
- 3) 함수 호출과 반환
- 4) 함수와 변수의 종류
- 5) 함수에 배열 전달
- 6) 함수와 라이브러리

6) 함수와 라이브러리

- **라이브러리**
 - 함수들을 구현해 모아 놓은 것
 - 필요 시 함수를 호출하여 사용
- **표준 라이브러리**
 - C언어에서 정해놓은 표준 함수들로 구성: printf(), scanf() 등
- **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>