
컴퓨터 프로그래밍 개론

함수 사용과 변수의 수명

학습목차

- I. 함수 개요
- II. 함수의 작성과 사용
- III. 여러 가지 유형의 함수
- IV. 변수의 수명

함수 개요

▶ 함수 개요

- ▶ 특정 작업을 수행하는 명령어의 모음
- ▶ 함수를 사용하면 복잡한 처리과정을 효율적으로 수행 가능
- ▶ C언어로 작성한 프로그램은 함수들의 집합체
 - ▷ 함수들끼리 서로 불러서(호출)하여 사용하는 형태로 실행

```
#include <stdio.h>

double CircleOfArea(double r) {
    double result;

    if (r < 0) {
        r *= -1;
    }
    result = r * r * 3.14;

    return result;
}
```

```
int main() {
    double inputNum;
    double result;

    printf("반지름을 입력 : ");
    scanf("%lf", &inputNum);

    result = CircleOfArea(inputNum);

    printf("원의 넓이는 %f\n", result);
    return 0;
}
```

함수 개요

▶ 함수의 종류

▶ 표준 라이브러리 함수

- ▶ 프로그래밍 언어에서 제공하는 함수
- ▶ 자주 사용하는 기능을 미리 작성한 후 호출하여 사용

▶ 사용자 정의 라이브러리 함수

- ▶ 사용자가 직접 작성하는 함수

▶ 함수 사용의 장점

▶ 코드의 안정성 향상

- ▶ 모듈화를 통하여 코드의 의존성을 낮추고 함수의 독립성을 높임

▶ 에러 수정이 쉬움

- ▶ 함수가 독립적으로 존재하면 오류의 발견, 수정이 용이

▶ 재사용성 향상

- ▶ 다른 프로젝트에 활용 가능

```
void IsEvenOdd(int num){  
    if(num%2==0){  
        printf("Even Number\n");  
    }else{  
        printf("Odd Number\n");  
    }  
}
```

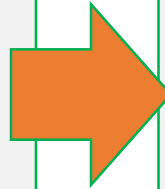
```
int TriangleArea(int base, int height){  
    return base * height / 2;  
}
```

함수의 필요성

▶ 함수를 사용하지 않는 경우

▶ 덧셈 과정에서 피연산자가 음수일 경우에 양수로 변환하여 계산

```
int main() {  
    int var1 = 1, var2 = -3, var3 = 5;  
    int var4 = -2, var5 = 7, var6 = -9;  
    int result1, result2, result3;  
  
    result1 = var1 + var2;  
    result2 = var3 + var4;  
    result3 = var5 + var6;  
  
    return 0;  
}
```



```
int main() {  
    int var1 = 1, var2 = -3, var3 = 5;  
    int var4 = -2, var5 = 7, var6 = -9;  
    int result1, result2, result3;  
    if(var1 < 0) var1 *= -1;  
    if(var2 < 0) var2 *= -1;  
    result1 = var1 + var2;  
    if(var3 < 0) var3 *= -1;  
    if(var4 < 0) var4 *= -1;  
    result2 = var3 + var4;  
    if(var5 < 0) var5 *= -1;  
    if(var6 < 0) var6 *= -1;  
    result3 = var5 + var6;  
  
    return 0;  
}
```

함수의 필요성

▶ 함수를 활용하는 경우

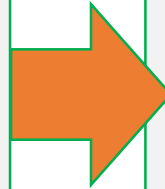
▶ 덧셈 과정에서 피연산자가 음수일 경우에 양수로 변환하여 계산

```
int add(int a, int b){
    return a + b;
}

int main() {
    int var1 = 1, var2 = -3, var3 = 5;
    int var4 = -2, var5 = 7, var6 = -9;
    int result1, result2, result3;

    result1 = add(var1, var2);
    result2 = add(var3, var4);
    result3 = add(var5, var6);

    return 0;
}
```



```
int add(int a, int b){
    if(a < 0) a *= -1;
    if(b < 0) b *= -1;
    return a + b;
}

int main() {
    int var1 = 1, var2 = -3, var3 = 5;
    int var4 = -2, var5 = 7, var6 = -9;
    int result1, result2, result3;

    result1 = add(var1, var2);
    result2 = add(var3, var4);
    result3 = add(var5, var6);

    return 0;
}
```

함수의 필요성

▶ 함수의 장점 정리

▶ 코드의 안정성이 높아지고 오류를 수정하기 편리

▷ 기능별로 함수를 분류하였으므로 오류를 쉽게 찾아내고 함수 단위로 수정할 수 있음

▷ 또한 함수를 많이 사용할수록 코드의 독립성과 안정성이 높아짐

▶ 코드의 재사용(재활용)

▷ 미리 작성해 놓은 함수를 필요할 때마다 호출하여 사용 가능

▷ 프로그램을 작성하는 과정에서 재사용이 잘 되도록 함수를 작성해야 함

▷ 함수로 재사용하는 것을 부품화라고 하며 다양한 부품이 준비되어 있으면 복잡한 프로그램도 부품을 조합하는 것만으로 완성 가능

▷ 부품을 재사용 할 수록 프로그램 개발의 시간을 절약

함수의 작성과 사용 방법

▶ 함수의 세가지 형태

▶ 함수는 프로그램에서 선언, 정의, 호출의 세 가지 상태로 사용

▶ 선언

▷ 함수의 형태를 컴파일러에게 알림

▷ 함수 원형에 세미콜론;을 붙임

▶ 정의

▷ 함수를 사용할 경우 동작할 코드를 작성

▷ 반환 값의 형태, 이름, 매개변수를 표시하고 블록{} 내부에 기능을 구현

▶ 호출

▷ 작성된 함수를 사용할 경우 함수의 이름과 필요한 인자를 전달

선언	<code>int add (int a, int b);</code>
정의	<code>int add (int a, int b)</code> <code>{</code> <code> return a + b ;</code> <code>}</code>
호출	<code>add (10, 20);</code>

함수의 작성과 사용 방법

▶ 함수 원형(function prototype)

- ▶ 함수이름, 필요한 데이터, 결과값을 한 줄에 요약한 것

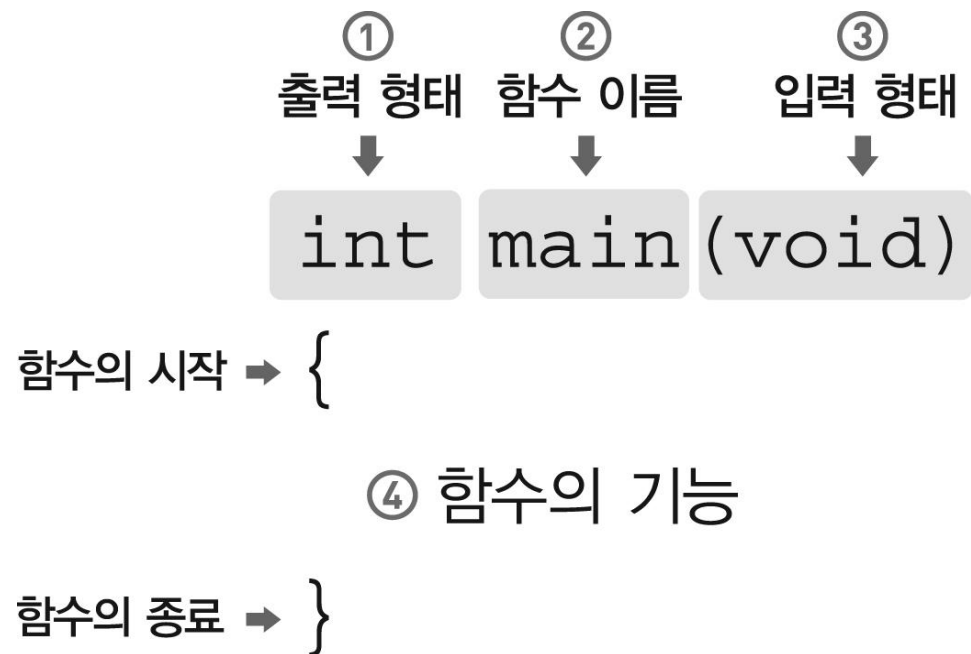
▶ 함수의 구성요소

- ▶ **출력 형태** : 함수의 출력 형식(자료형)을 나타냄

- ▶ 함수 이름 : 함수의 고유한 이름을 표현

- ▶ **입력 형태** : 함수가 입력 받을 인수의 자료형과 개수를 정의

- ▶ 함수의 기능 : 함수가 수행할 기능 정의



함수의 작성과 사용 방법

▶ 함수명 작성 방법

- ▶ 첫 글자는 대문자로 작성하고 단어 중간에 언더바_를 사용하지 않음
- ▶ 나머지는 변수 이름 짓는 법과 동일
 - ▷ 영문자(a~z, A~Z), 숫자(0~9), 밑줄을 조합하여 구성
 - ▷ 숫자로 시작하는 이름은 오류 발생
 - ▷ 공백 포함 불가
 - ▷ C언어 문법에서 이미 정의한 예약어를 단독으로 사용하여 함수명을 작성 불가
 - ▷ void, return, char, int, ...
 - ▷ void int(int void){ ...

함수의 작성과 사용 방법

▶ 함수의 두 가지 사용방법

▶ 첫 번째 방법

- 1) 함수의 정의
- 2) 함수의 호출(함수의 사용)

▶ 두 번째 방법

- 1) 함수의 선언
- 2) 함수의 호출(함수의 사용)
- 3) 함수의 정의

함수의 작성과 사용 방법

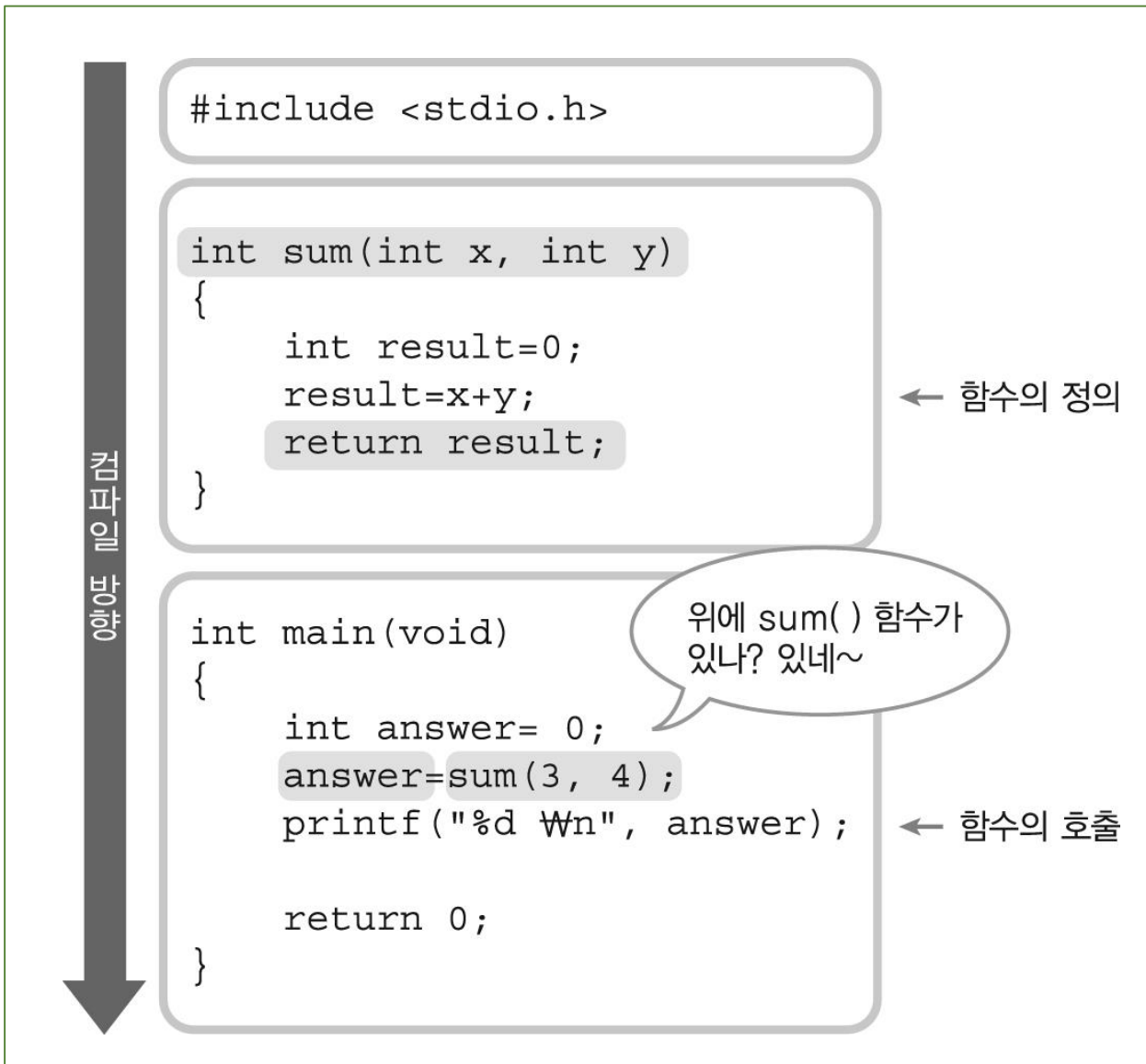
▶ 함수 사용 방법 - 1

A. 함수의 정의

- ▶ 함수의 기능을 정의한 문장

B. 함수의 호출

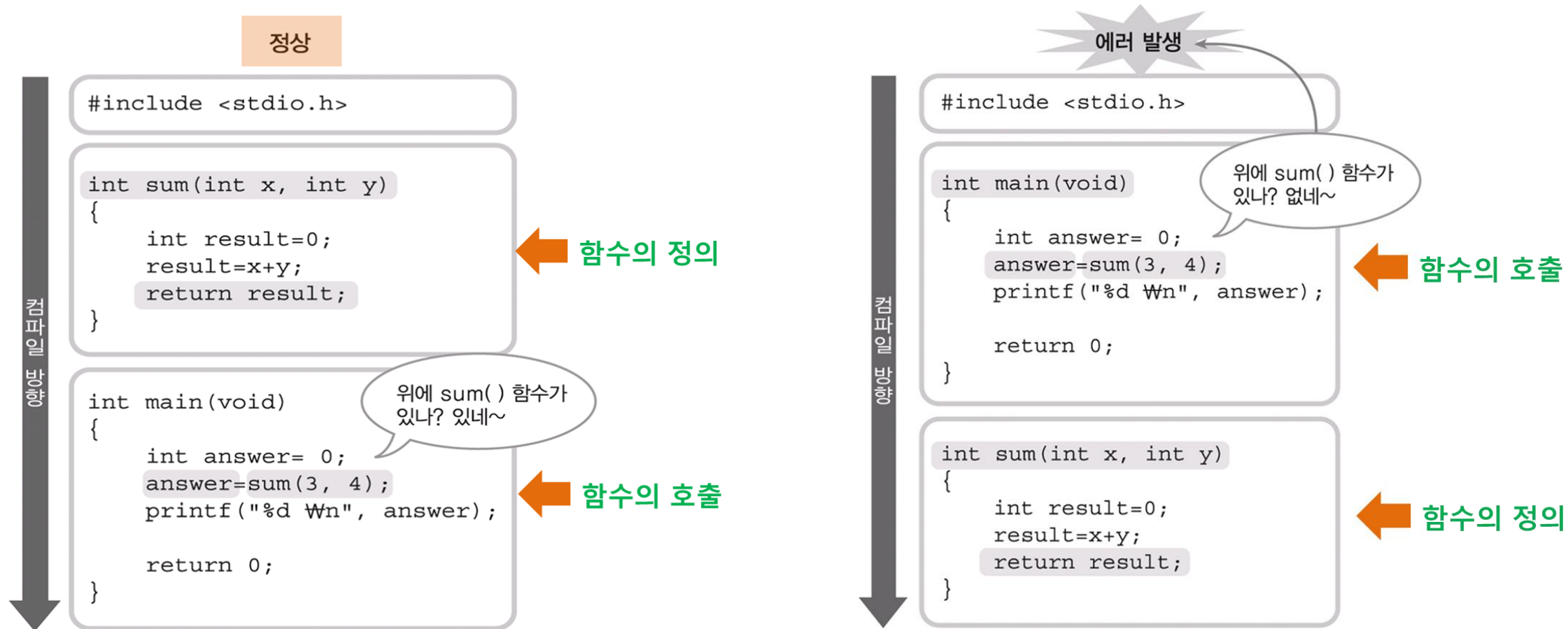
- ▶ 정의된 함수를 호출 하는 문장



함수의 작성과 사용 방법

▶ 함수 사용 방법 - 1

▶ 소스 코드는 위에서 아래로 컴파일 되므로 함수를 호출하기 전에 미리 정의하여 컴파일러가 해당 함수의 정보를 알 수 있도록 함



함수의 작성과 사용 방법

▶ 함수 사용 방법 - 1

- ▶ 함수가 많아지면 항상 함수의 **호출 순서에 주의**해서 작성해야 하는 번거로움이 있음

```
void printResult(int i){  
    printf("결과 %d\n",i);  
}  
  
void sum(int a, int b){  
    printResult(a+b);  
}  
  
int main(void) {  
    sum(2,3);  
    return 0;  
}
```

```
void sum(int a, int b){  
    printResult(a+b); //error  
}  
  
void printResult(int i){  
    printf("결과 %d\n",i);  
}  
  
int main(void) {  
    sum(2,3);  
    return 0;  
}
```

함수의 작성과 사용 방법

▶ 함수 사용 방법 - 2

A. 함수의 선언

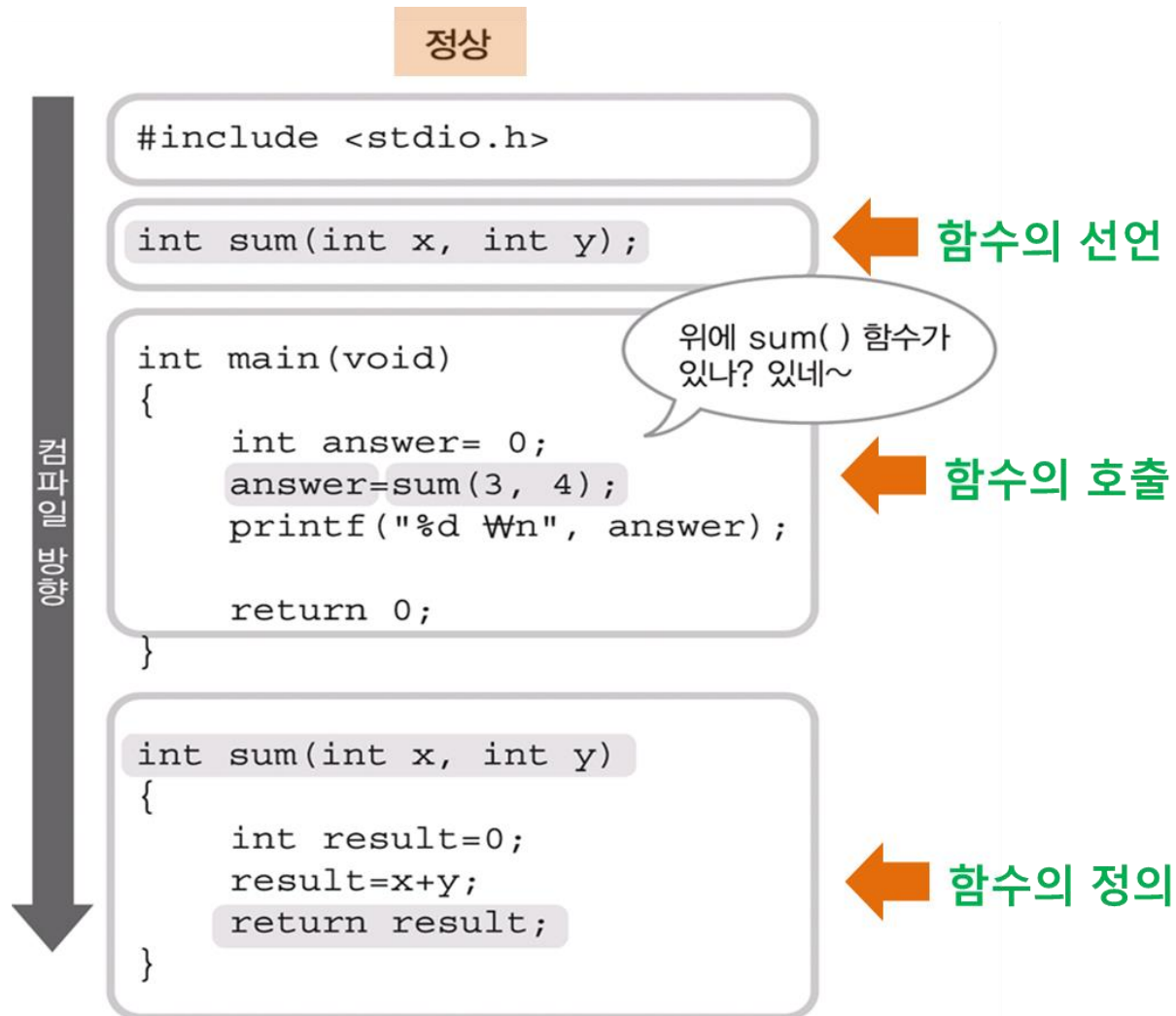
- ▶ 함수의 목록이 있는 문장

B. 함수의 호출

- ▶ 정의한 함수를 호출 하는 문장

C. 함수의 정의

- ▶ 함수의 기능을 정의한 문장



함수의 작성과 사용 방법

▶ 함수의 선언

- ▶ 함수를 프로그래밍하는 일반적인 방법
 - ▷ main 함수만 프로토 타입이 필요하지 않음
- ▶ 소스 코드의 상단에서 함수 목록들을 직관적으로 확인 가능
- ▶ 대략적으로 함수의 기능 분석 가능
- ▶ 호출 순서를 고려할 필요가 없음

▶ 선언 방법

- ▶ 함수 원형에 세미콜론(;) 붙임
- ▶ 선언하는 위치는 main 함수 위
- ▶ 매개변수 이름 생략 가능
 - ▷ 그러나 매개변수 이름을 생략하면 프로그래머가 원형만 보고 함수의 기능을 예측하기 어려움 없음

```
void sum(int, int);  
void printResult(int result);  
int main(void) {  
    sum(2, 3);  
    return 0;  
}  
void sum(int a, int b){  
    printResult(a+b);  
}  
void printResult(int i){  
    printf("결과 %d\n",i);  
}
```


함수의 작성과 사용 방법

▶ 함수 선언이 필요한 이유

▶ 함수 선언에서 반환 값 형태 확인

▷ 함수가 종료될 때 아무런 반환 값이 없는 구조일 경우 타입은 void

▶ 함수를 호출하기 전에 함수의 선언을 통하여 **반환형을 미리 컴파일러에게 알림**

▷ 컴파일러는 함수 호출 시 반환 값과 같은 형태의 저장 공간 준비할 필요가 있음

▷ `answer = sum(3, 4);`

▷ 함수 정의에서도 반환형 확인 가능

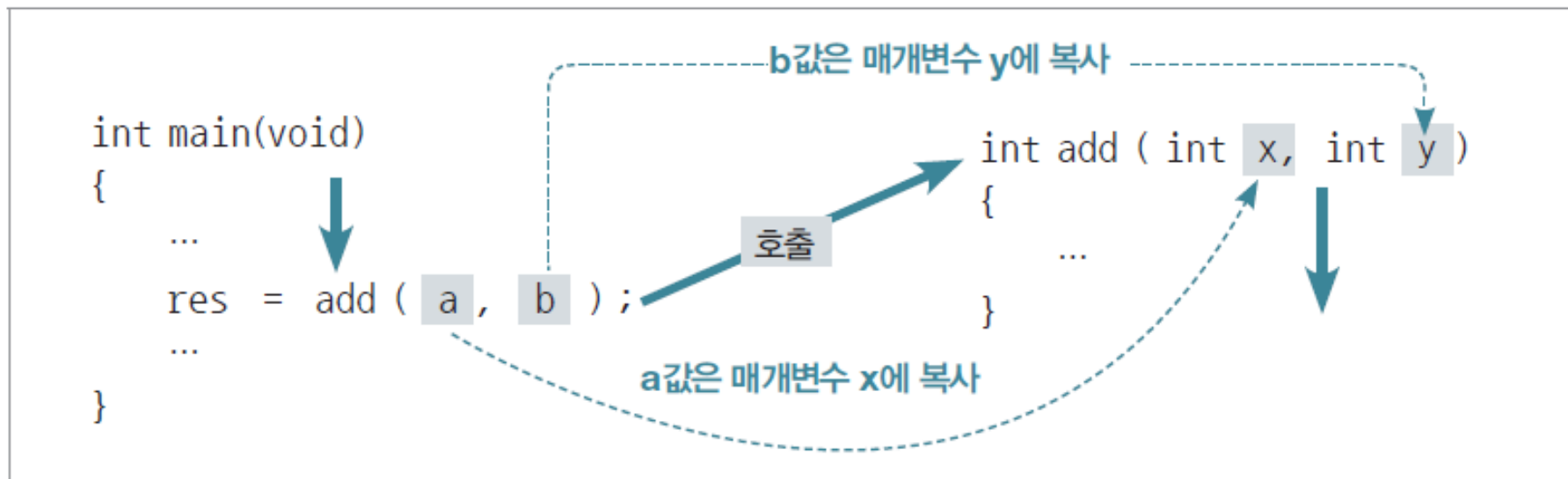
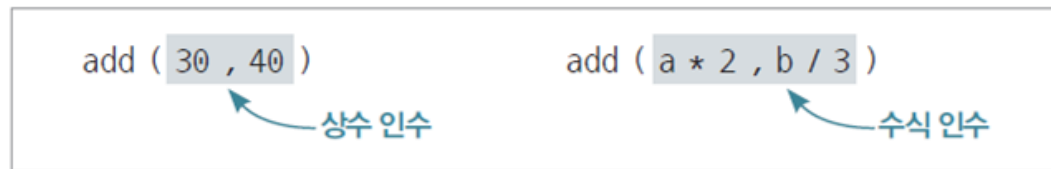
▶ 함수 호출형식에 문제가 없는지 검사

▷ 호출할 때 정확한 인자를 주는지 검사 가능

함수의 작성과 사용 방법

▶ 함수의 호출

- ▶ 함수는 호출 통하여 실행
- ▶ 함수를 호출할 때는 함수에 필요한 인수를 괄호 안에 넣어줌
- ▶ 인수는 함수의 매개변수에 **순서대로 복사**
- ▶ 인수로 상수나 변수를 사용
 - ▷ 인수로 수식을 입력하면 계산된 결과값을 인수로 전달



함수의 작성과 사용 방법

▶ 가인수와 실인수의 차이

▶ 가인수(Parameter)

- ▶ 함수 내부에서 사용하기 위하여 정의된 변수

```
int sum(int a, int b);
```

▶ 실인수(Argument)

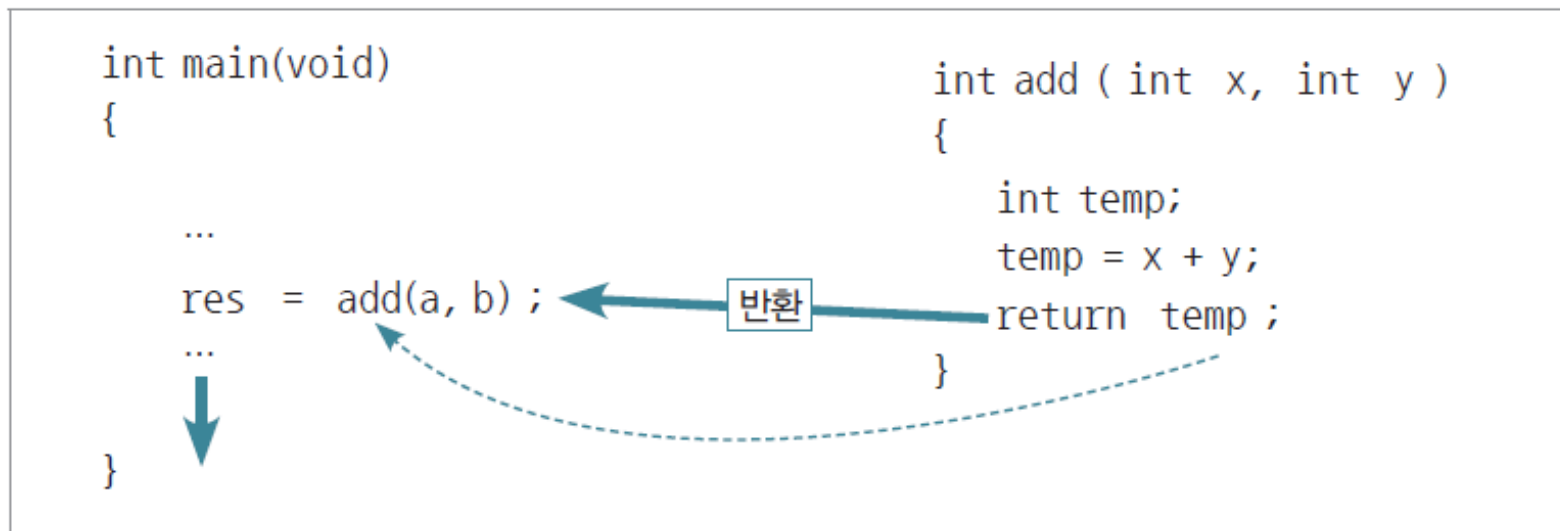
- ▶ 함수를 호출하여 내부적으로 처리하도록 전달하는 데이터
- ▶ 가인수를 가진 함수를 호출할 경우에는 반드시 실인수를 입력하여 전달해야 함

```
int result = 0;  
result = sum(2, 3);  
result = sum(2);           //오류 발생 : 가인수와 실인수의 개수가 동일해야 함  
result = sum( );          //오류 발생 : 가인수를 정의한 경우 반드시 실인수 입력
```

함수의 작성과 사용 방법

▶ 함수 호출과 반환

- ▶ 함수를 종료하거나 함수의 실행을 끝내고 호출한 곳으로 돌아가기 위해 return문 사용
 - ▷ 반환 값은 함수로부터 반환된 계산 결과를 전달 받거나 실행 과정에서 발생하는 오류 여부를 확인할 경우 사용
- ▶ 컴파일러는 함수를 호출할 때 반환 값의 저장 공간을 미리 준비
 - ▷ 함수를 선언하거나 정의할 때 입력한 함수 타입과 동일한 데이터가 실행 결과로 반환됨
- ▶ 인수는 여러 개를 전달할지라도 반환 값은 반드시 한 개
 - ▷ 아무것도 호출한 곳으로 반환하지 않으려면 함수의 타입을 void로 지정



함수의 작성과 사용 방법

▶ 사칙연산관련 함수 사용 예제

▶ 두 개의 피연산자를 입력 받고 몫과 나머지를 구하는 프로그램

```
#include <stdio.h>
int divide(int x, int y);
int input(void);
void output(int x);
void information(void);
int mod(int, int);
int main(void)
{
    int num1, num2, result, remainder;
    information();
    printf("첫 번째 정수 입력: ");
    num1 = input();
    printf("두 번째 정수 입력: ");
    num2 = input();
    result = divide(num1, num2);
    remainder = mod(num1, num2);
    output(result, remainder);
    return 0;
}
```

함수의 작성과 사용 방법

▶ 사칙연산관련 함수 사용 예제(계속)

▶ 두 개의 피연산자를 입력 받고 몫과 나머지를 구하는 프로그램

```
int mod(int x, int y) {
    int value;
    if (y == 0) {
        printf("나머지 계산 오류!\n");
        y = 1;
    }
    value = x % y;
    return value;
}

int divide(int x, int y) {
    int value;
    if (y == 0) {
        printf("나누기 계산 오류!\n");
        y = 1;
    }
    value = x / y;
    return value;
}
```

```
int input(void) {
    int value;
    scanf_s("%d", &value);
    return value;
}

void output(int x, int y) {
    printf("나눗셈 결과: %d, 나머지 : %d \n", x, y);
    return;
}

void information(void) {
    printf("--- 프로그램 시작 ---\n");
    return;
}
```

실습예제 1

▶ 정수와 사칙연산 기호를 입력 받아 연산을 하는 코드를 작성하시오

▶ 연산은 함수를 이용하여 동작

▷ 나누기 연산에서 나머지는 출력하지 않음

▶ 피연산자 2개를 먼저 입력 받고 연산자를 입력 받음

▶ 함수 이름

▷ add(int, int), sub(int, int), mul(int, int)

▶ 실행결과

```
피연산자와 사칙연산자가 포함된 수식을 입력 : 10 20 +  
10 + 20 = 30
```

여러 가지 함수 유형

▶ 재귀호출 함수(Recursive Function)

▶ 함수 내부에서 자기 자신을 호출하는 함수

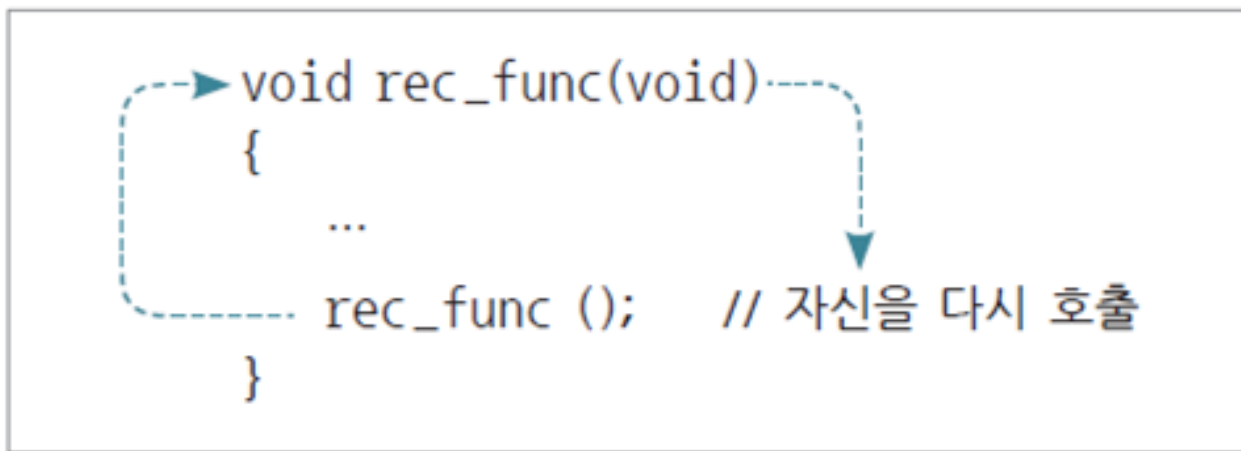
▷ 자료구조나 알고리즘 분야에서 복잡한 문제들을 해결할 경우 사용

▶ 재귀 호출(Recursive Call) : 자기 자신을 호출하는 행위

▶ 재귀 호출의 문제점

▷ 계속적인 자기 호출로 시간과 메모리 공간의 효율이 저하

▷ 개발에 신중해야 함



```
void rec_func(void)
{
    ...
    rec_func ();    // 자신을 다시 호출
}
```

The diagram illustrates a recursive function call. It shows a C++ function definition for `void rec_func(void)`. Inside the function body, there is an ellipsis (`...`) followed by the line `rec_func ();` with a comment `// 자신을 다시 호출` (// call myself again). A dashed blue arrow originates from the function name `rec_func` in the function signature, loops around the opening curly brace, and points down to the `rec_func ();` line, indicating the function calling itself.

여러 가지 함수 유형

▶ 재귀호출 함수 예제

▶ 무한 루프

```
#include <stdio.h>
void self_service(void);
int main(void)
{
    self_service( );
    return 0;
}

void self_service(void) {
    printf("셀프서비스\n");
    self_service( );
}
```

여러 가지 함수 유형

▶ 재귀호출 함수 예제

▶ 5회만 실행되도록 수정

```
#include <stdio.h>
void self_service(void);
int main(void)
{
    self_service( );
    return 0;
}

void self_service(void)
{
    static int i=1;          // int i=1;
    if(i>5)                  // 함수의 '무한 반복 문제'를 해결하는 조건
        return;              // 값을 반환하지 않고 그냥 함수를 종료

    printf("셀프서비스 %d 회 \n", i);
    i=i+1;
    self_service( );
}
```

여러 가지 함수 유형

▶ 재귀호출 함수 예제

▶ 매개 변수를 이용한 재귀 호출

```
#include <stdio.h>
void self_service(int n);

int main(void){
    int a = 1;
    self_service(a);
    return 0;
}

void self_service(int n){
    if (n > 5)
        return;

    printf("셀프서비스 %d 회 \n", n);
    self_service(n + 1);    // n을 하나 증가해서 self_service( ) 함수 재귀 호출
}
```

여러 가지 함수 유형

▶ 재귀호출 함수 예제

▶ return 문을 이용한 재귀 호출

```
#include <stdio.h>
int factorial(int n);
int main(void){
    int a;
    int result;
    printf("정수입력: " );
    scanf("%d", &a);

    result=factorial(a);
    printf( "%d 팩토리얼은: %d입니다. \n", a, result);
    return 0;
}
int factorial(int n){
    if (n<=1)
        return 1;
    else
        return n * factorial(n-1);
}
```

실습예제 2

▶ 1부터 사용자가 입력한 정수 n 까지의 합을 구하는 재귀함수를 작성하시오.

▶ 3을 입력하면 6이 출력되고 5를 입력하면 15가 출력됨

변수의 수명

▶ 지역 변수(local variable)

- ▶ 변수가 선언된 블록{ } 내부(scope)에서만 데이터가 유효
- ▶ 함수 내부에서 선언된 변수는 함수가 종료되면 메모리에서 사라짐
 - ▶ 함수가 아니더라도 블록{ }이 끝나면 내부에 선언된 변수들은 모두 메모리에서 사라짐
- ▶ 같은 이름의 지역 변수라도 서로 다른 함수에서 선언되었다면 별개임
 - ▶ 함수의 독립성으로 인하여 지역변수의 이름이 같을지라도 서로 다른 메모리에 위치함
 - ▶ 따라서 예제의 count 변수는 서로 관계가 없음
 - ▶ 지역 변수를 main함수에 선언하면 main함수가 종료될 때까지는 사용 가능

```
void countfunc();

int main(void) {
    int count = 1;
    countfunc();
    countfunc();
    countfunc();
    return 0;
}

void countfunc(){
    int count = 0;
    count++;
    printf("%d\n",count);
}
```

변수의 수명

▶ 지역 변수의 유효 범위를 확인할 수 있는 예제

```
int main(){
    int value1 = 10;
    int value2 = 20;
    printf("1:value1 %d\n", value1);
    printf("1:value2 %d\n", value2);
    {
        int value1 = 30;
        value2 = 40;
        printf("2:value1 %d\n", value1);
        printf("2:value2 %d\n", value2);
    }
    printf("3:value1 %d\n", value1);
    printf("3:value2 %d\n", value2);

    return 0;
}
```

변수의 수명

▶ 전역 변수(global variable)

- ▶ 함수 밖에서 선언한 변수이며 어떠한 함수에서도 자유롭게 사용 가능
 - ▷ 여러 함수가 공유하여 사용
- ▶ 프로그램이 종료될 때까지 살아남아 선언된 소스 파일 안의 모든 함수에서 사용
 - ▷ 프로그램이 시작될 때 생성되어 종료될 때 프로그램과 함께 사라짐
- ▶ 초기화할 경우 반드시 리터럴 상수로 초기화해야 함
 - ▷ 프로그램 시작 시에 처음 한번만 초기화
 - ▷ 전역변수는 별도로 초기화를 하지 않으면 0으로 자동 초기화
- ▶ 협업으로 개발할 경우, 주의해서 사용
 - ▷ 잘못 사용하면 프로그램의 수정, 유지 보수, 재사용을 어렵게 함

```
const int NUM = 0;
int num = NUM;    //오류 발생
int count;        //0으로 초기화
void countfunc();
int main(void) {
    int count = 10;
    countfunc();
    countfunc();
    countfunc();
    return 0;
}
void countfunc(){
    count++;
    printf("%d\n",count);
}
```


변수의 수명

▶ 전역 변수와 지역 변수의 우선 순위

- ▶ 함수의 독립성을 위하여 전역 변수와 동일한 이름의 지역 변수가 선언되면 해당 블록 내부에서는 지역 변수가 우선

```
int count;
void countfunc();
int main(void) {
    int count = 5;
    countfunc();
    count = 10;
    countfunc();
    countfunc();
    printf("%d\n", count);
    return 0;
}
void countfunc(){
    count++;
    printf("%d\n", count);
}
```

변수의 수명

▶ 정적 지역 변수(정적 변수 / static 변수)

- ▶ 변수의 타입 앞에 static을 붙임
- ▶ 지역 변수와 전역 변수의 특징을 모두 가지고 있는 변수
 - ▷ 지역 변수처럼 함수 내부에 선언
 - ▷ 전역 변수처럼 프로그램이 종료될 때 까지 메모리에 남아 있음
 - ▷ 함수의 호출과 무관하게 프로그램이 실행되면 메모리에 할당
 - ▷ 이전에 함수가 호출되었을 때 값을 기억하기 원하는 경우에 사용
 - ▷ 정적 지역 변수가 선언된 함수 외부에서 해당 변수에 접근할 경우 오류 발생
 - ▷ 정적 지역 변수를 전역 변수와 동일한 방식으로 접근 불가
 - ▷ 모든 함수에서 수정 및 접근 가능한 전역 변수의 단점을 부분적으로 보완
 - ▷ 전역 변수와 마찬가지로 리터럴 상수로만 초기화를 할 수 있음
 - ▷ 따로 초기화 하지 않는 경우 0으로 자동 초기화
 - ▷ 어떠한 상황에서도 초기화는 프로그램이 시작될 때 단 한번만 수행됨

```
void countfunc();
int main(void) {
    countfunc();
    countfunc();
    countfunc();
    return 0;
}
void countfunc(){
    static int count;
    count++;
    printf("%d\n",count);
}
```

실습예제 3

▶ 두 정수를 입력 받아 덧셈을 계산하는 함수 sum 함수를 작성하시오.

▶ 총 연산은 3번으로 제한하며 두 정수 모두 0을 입력하면 종료

▶ 연산 개수의 카운트는 sum함수 내부에 정적 변수로 작성

▶ 입력 값을 콤마,로 구분하여 입력 받도록 작성

▶ 실행결과1

```
정수 두개를 입력하세요(0 0->exit) : 3,3
덧셈결과 : 6
정수 두개를 입력하세요(0 0->exit) : 0,0
프로그램 종료!!
```

▶ 실행결과2

```
정수 두개를 입력하세요(0 0->exit) : 10,10
덧셈결과 : 20
정수 두개를 입력하세요(0 0->exit) : 20,20
덧셈결과 : 40
정수 두개를 입력하세요(0 0->exit) : 30,10
덧셈결과 : 40
프로그램 종료!!
```

실습예제 4

▶ 알파벳 문자 한 개를 입력 받아서 알파벳 순서로 몇 번째 문자인지 출력하는 프로그램을 작성하시오.

▶ 알파벳 순서를 판단하는 alpha함수를 작성

▶ hint

▷ char 형도 연산이 가능

▷ 아스키 코드 값 : A - 65, a - 97

Q & A
