-Part1-

제9장 함수란 무엇인가

학습목차

9.1 함수란

9.2 다양한 형태의 함수들

9.3 함수 적용 방법

9.4 변수의 종류와 범위

9.5 재귀 함수

9.1 함수란

9.1 함수란

▶ 함수

✓특정 작업을 수행하는 **코드의 집합**

▶ 함수의 종류

✓**표준 라이브러리 함수** C 언어에서 제공

✓**사용자 정의 라이브러리 함수** 사용자가 직접 만든 함수

▶ 함수 사용의 장점

✓코드의 **안정성** 향상

✓**에러 수정**이 쉬움

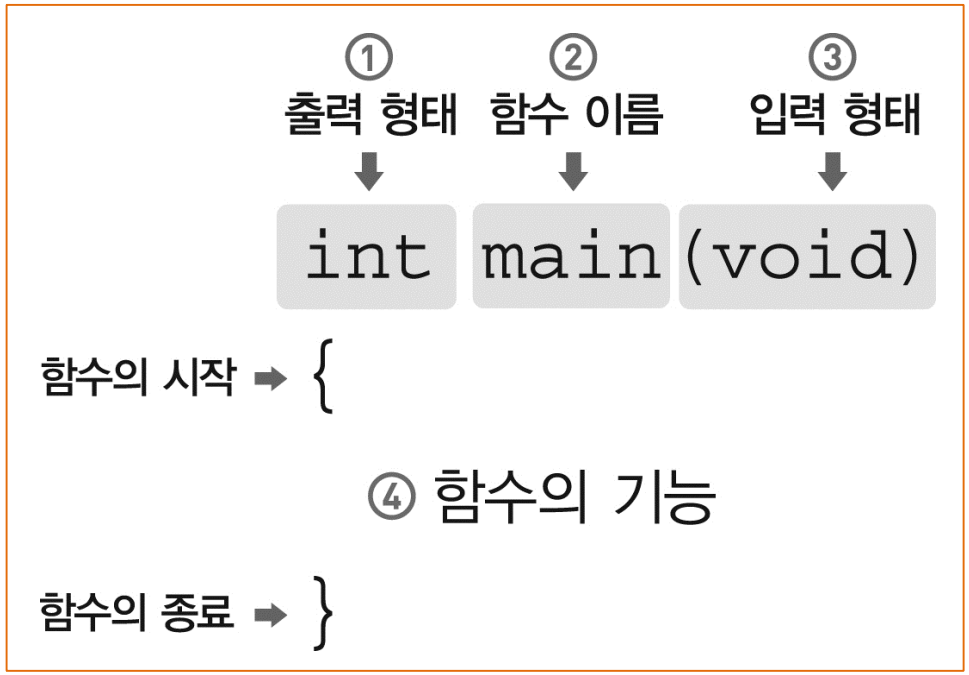
✓**재사용성** 향상

✓**복잡성↓, 응집력↑**

9.2 다양한 형태의 함수들

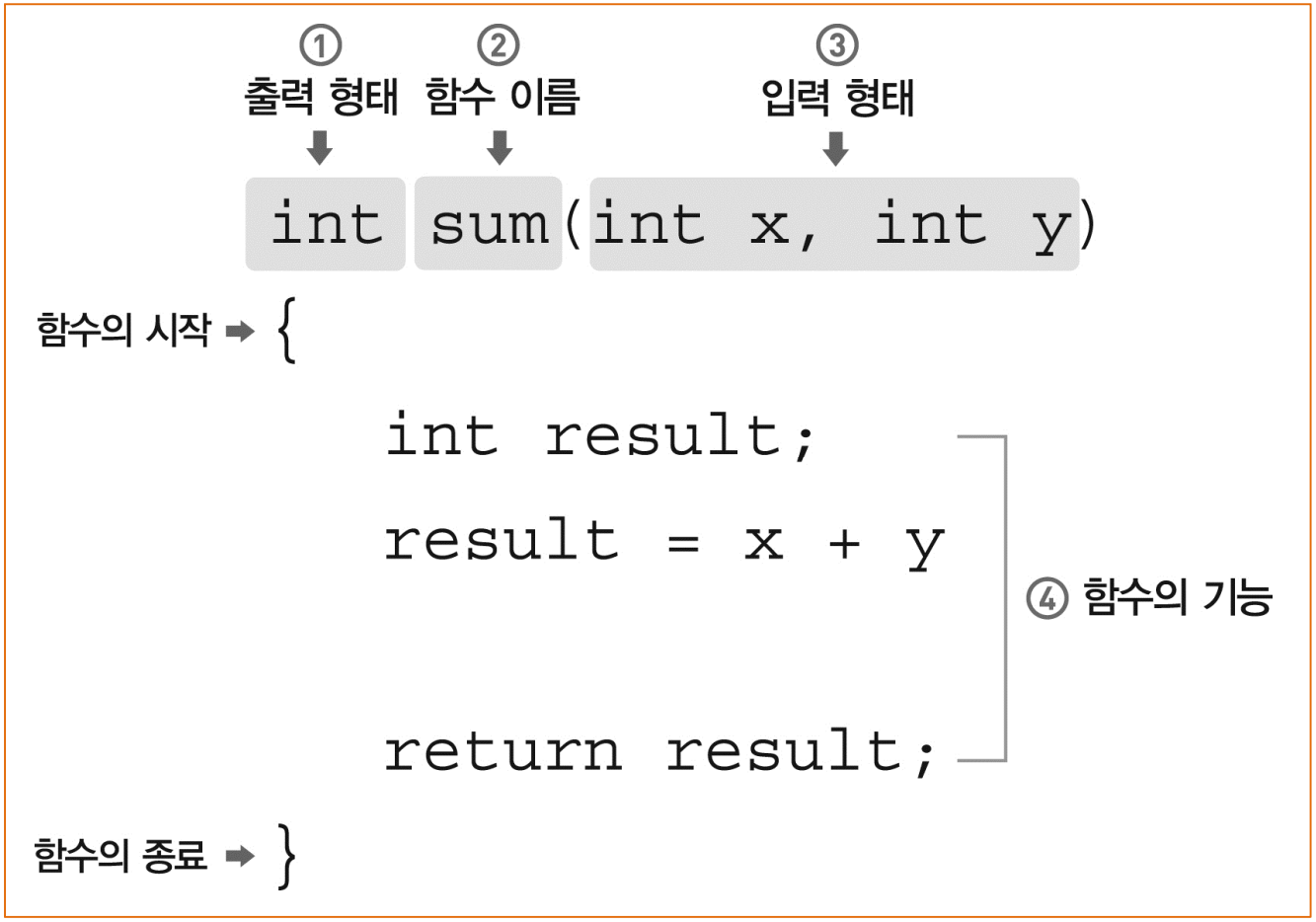
9.2 다양한 형태의 함수들 (1/7)

▶ 함수의 기본요소

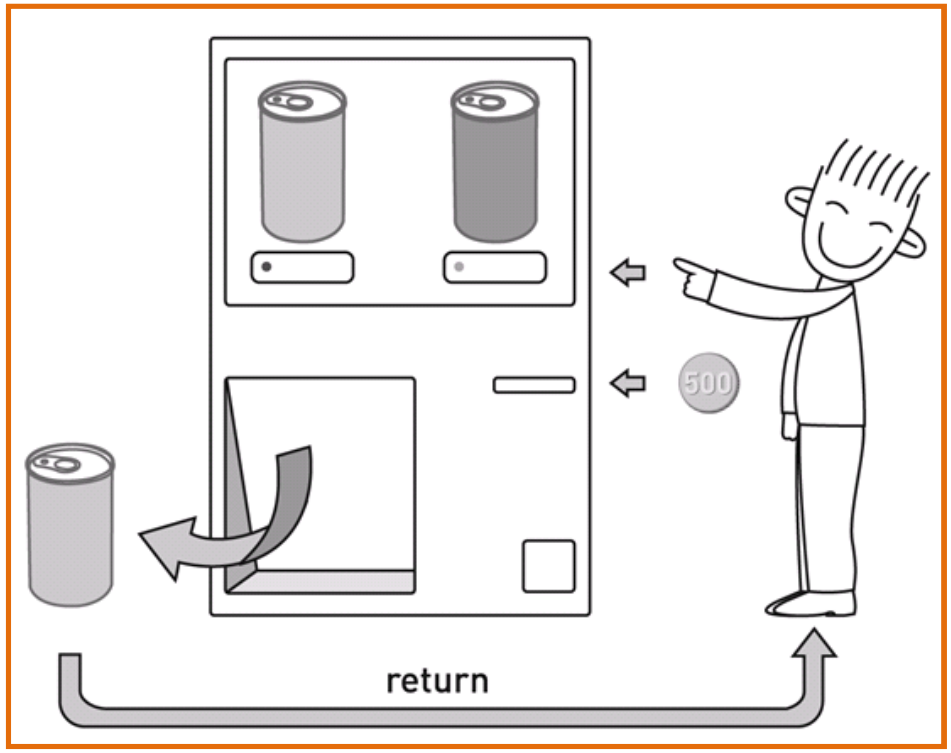


- **입력 형태 :** 함수가 입력 받을 형태
- **함수 이름 :** 함수의 이름을 표현
- **출력 형태 :** 함수의 출력을 나타내
- **함수의 기능 :** 함수가 수행할 기능 정의

9.2 다양한 형태의 함수들 (2/7)



사례: 음료 자판기




```
③ 출력 형태        ① 입력 형태
   ↓                  ↓
음료 자판기(동전, 버튼)
{
   ↑
② 함수 이름
    ...
    if(동전 && 버튼)
    return 콜라;
}
```

④ 함수의 기능

9.2 다양한 형태의 함수들 (3/7)---[9-1.c 실습]

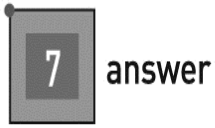
```
#include <stdio.h>
int sum(int x, int y)
{
        int result=0;
        result=x+y;
        return result;
}

int main(void)
{
        int answer = 0;
        answer=sum(3, 4);
        printf("%d ₩n", answer);

        return 0;
}
```

① 운영체제가 가장 먼저 main( ) 함수를 호출

② 3과 4를 가지고 sum( ) 함수를 호출해서 x에 3을 저장하고 y에 4를 저장

③ x+y의 결과인 7을 변수 result에 저장

④ result에 저장된 값 7을 변수 answer에 변환



⑤ main( ) 함수로 돌아와 남은 부분을 수행하고 프로그램을 종료

9.2 다양한 형태의 함수들 (4/7)

▶ 함수의 형태 4가지 – 11 형태

① 11 형태

```
int sum(int x, int y)
{
        int result=0;
        result=x+y;
        return result;
}
```

출력 형태	있음(int) ➔ 1
입력 형태	있음(int x, int y) ➔ 1
해석	x, y를 입력 받아 sum() 함수의 기능을 처리하고 int형으로 출력
특이점	출력 형태가 있어서 함수 내에서 반드시 return문을 사용해야 함

9.2 다양한 형태의 함수들 (5/7)

▶ 함수의 형태 4가지 – 10 형태

② 10 형태

```
int input(void)
{
        int num=0;
        scanf("%d", num);
        return num;
}
```

출력 형태	있음(int) ➔ 1
입력 형태	없음(void) ➔ 0
해석	입력 받는 값 없이 input() 함수의 기능을 처리하고 int형으로 출력
특이점	출력 형태가 있어서 함수 내에서 반드시 return문을 사용해야 함

9.2 다양한 형태의 함수들 (6/7)

▶ 함수의 형태 4가지 – 01 형태

③ 01 형태

```
void print(int x)
{
        int a=x;
        printf("%d", a);
        return;
}
```

출력 형태	없음(void) ➔ 0
입력 형태	있음(int x) ➔ 1
해석	값 하나를 입력받아 print() 함수의 기능을 처리하고 출력은 하지 않음
특이점	출력 형태가 void이므로 함수 내에서 return 문이 없어도 됨

9.2 다양한 형태의 함수들 (7/7)

▶ 함수의 형태 4가지 – 00 형태

④ 00 형태

```
void output(void)
{
        printf("Hello");
        printf("world");
        return;
}
```

출력 형태	없음(void) ➔ 0
입력 형태	없음(void) ➔ 0
해석	입력 없이 output() 함수의 기능을 처리하고 출력은 하지 않음
특이점	출력 형태가 void이므로 함수 내에서 return 문이 없어도 됨

9.3 함수 적용 방법

9.3 함수 적용 방법 (1/11)

▶ 함수 적용 방법 2가지

첫 번째 방법 → 함수의 정의 → 함수의 호출

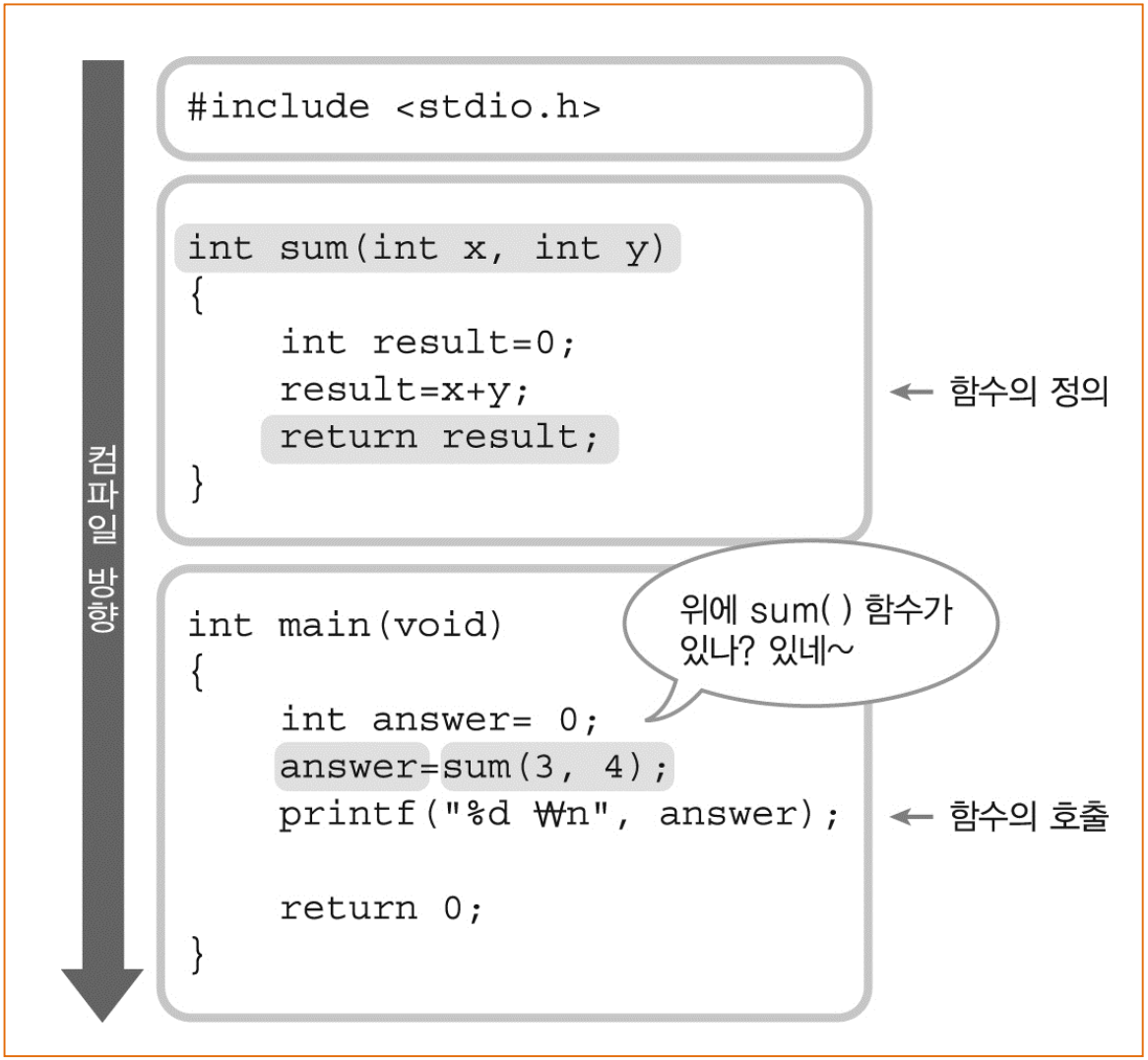
두 번째 방법 → 함수의 선언 → 함수의 호출 → 함수의 정의

9.3 함수 적용 방법 (2/11)

▶ 함수 적용 방법 1

✓**함수의 정의** 함수의 기능을 정의한 문장
✓**함수의 호출** 정의한 함수를 호출 하는 문장

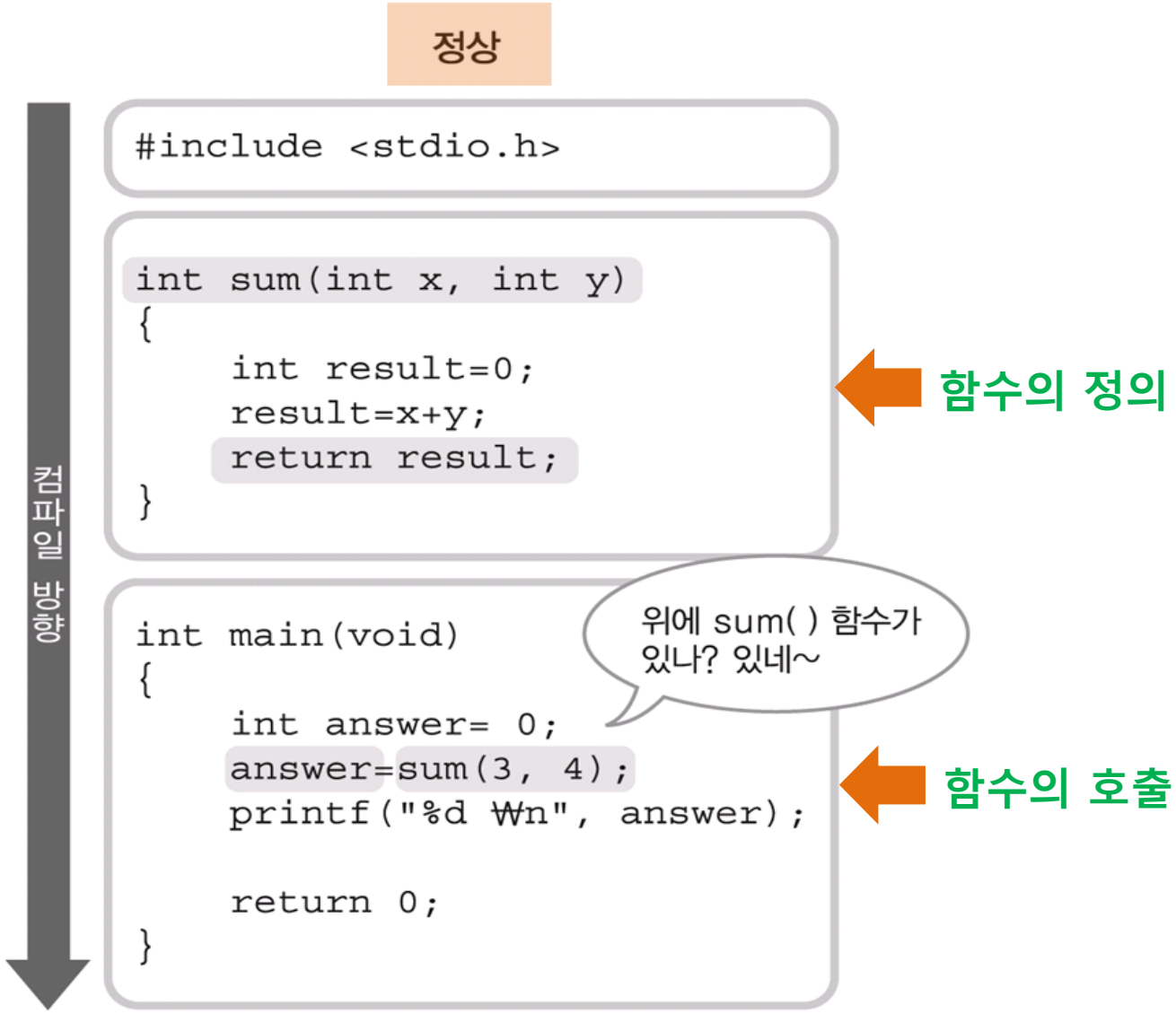
9.3 함수 적용 방법 (3/11)



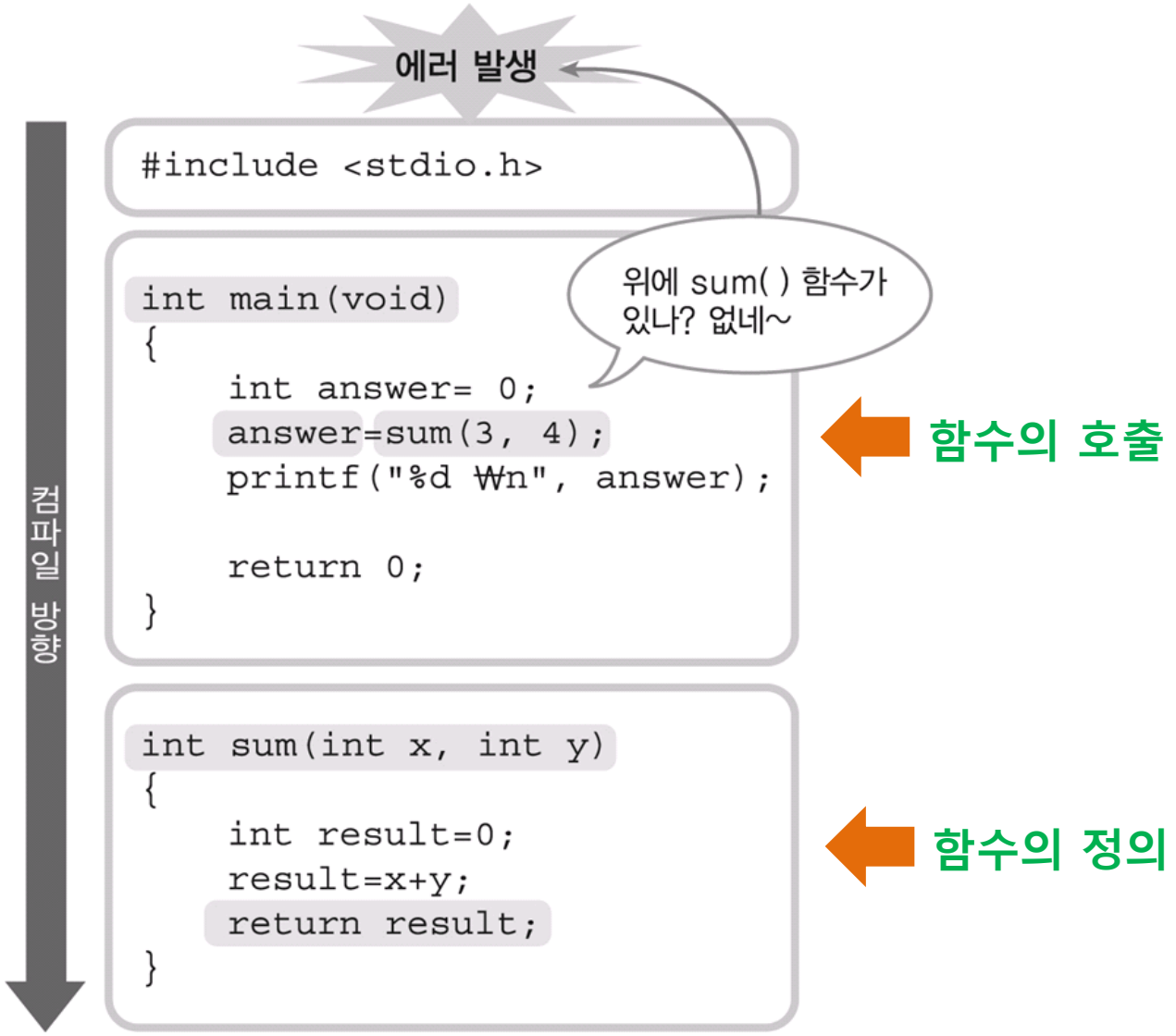
9.3 함수 적용 방법 (4/11)---[9-2.c 실습]

```
#include <stdio.h>
int max(int a, int b)                    // 함수의 정의(11 형태)
{
        if(a > b)
            return a;
        else
            return b;
}
int main(void)
{
        int i, j;
        int k;

        printf("숫자 두 개를 입력하세요: ");
        scanf("%d %d", &i, &j);

        k = max(i, j);                          //함수의 호출
        printf("%d와 %d 중 큰 수는 %d입니다. ₩n", i, j, k);
        return 0;
}
```

9.3 함수 적용 방법 (5/11)



9.3 함수 적용 방법 (6/11)

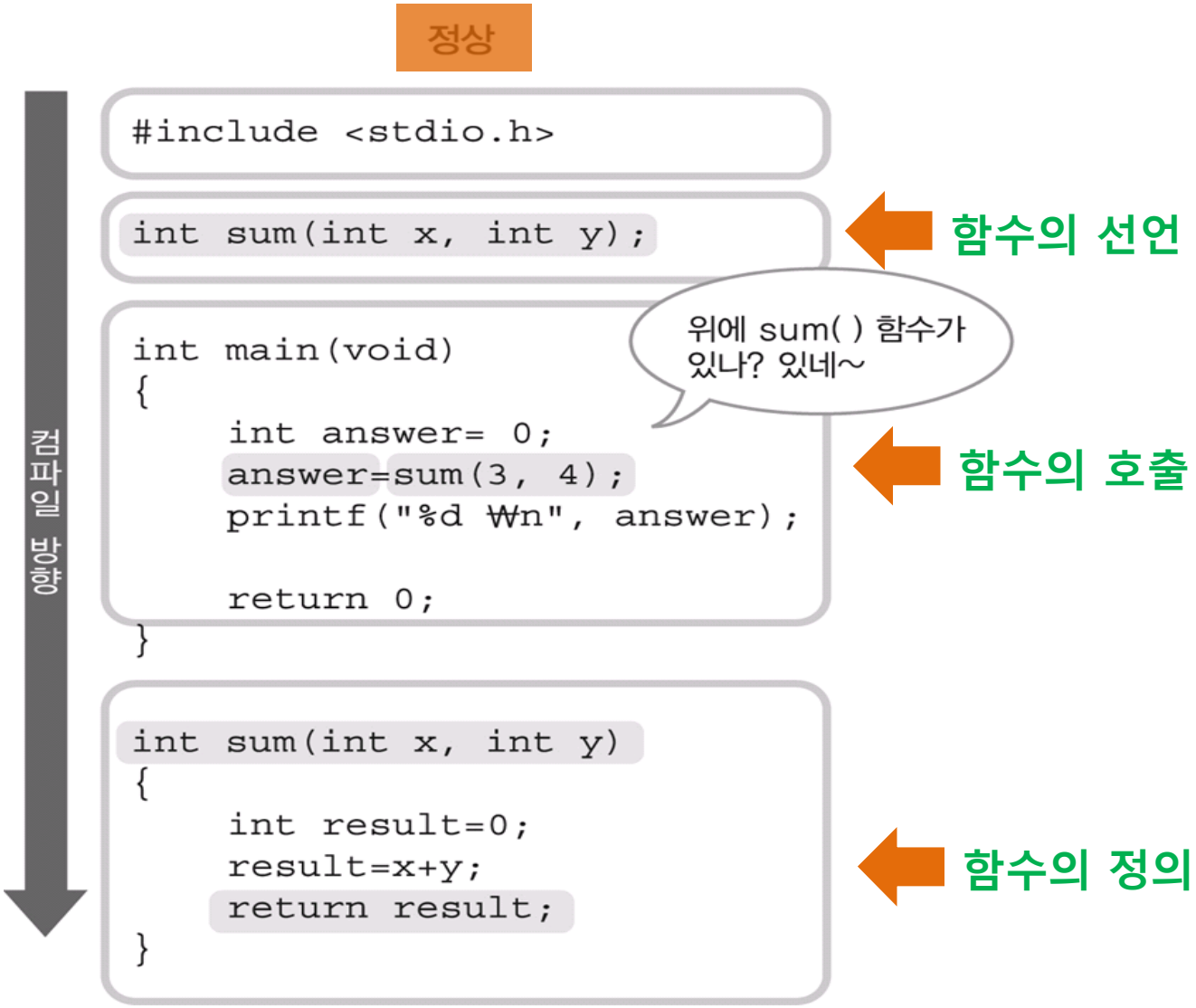


9.3 함수 적용 방법 (7/11)

▶ 함수 적용 방법 2

✓**함수의 선언** 함수의 목록이 있는 문장
✓**함수의 호출** 정의한 함수를 호출 하는 문장
✓**함수의 정의** 함수의 기능을 정의한 문장

9.3 함수 적용 방법 (8/11)



9.3 함수 적용 방법 (9/11)

▶ 함수의 선언

- ✓ 함수 적용에 있어서 **일반적인 방법**
- ✓ **'함수 목록들을 직관적으로 볼 수 있다.'**
- ✓ 대략적으로 함수의 기능 분석 가능

```
int sum(int x, int y);
```

9.3 함수 적용 방법 (10/11)---[9-3.c 실습(1/2)]

```
#include <stdio.h>
double divide(double x, double y);        // 함수의 선언(11 형태)
double input(void);                       // 함수의 선언(10 형태)
void output(double x);                    // 함수의 선언(01 형태)
void information(void);                   // 함수의 선언(00 형태)
int main(void)
{
        double num1, num2, result;

        information( );                   // 함수의 호출(00 형태)
        printf("첫 번째 실수 입력: ");
        num1=input( );                    // 함수의 호출(10 형태)

        printf("두 번째 실수 입력: ");
        num2=input( );                    // 함수의 호출(10 형태)

        result=divide(num1, num2);        // 함수의 호출(11 형태)
        output(result);
        return 0;
}
```

9.3 함수 적용 방법 (11/11)---[9-3.c 실습(2/2)]

```
double divide(double x, double y)          // 함수의 정의(11 형태)
{
        double val;
        val=x/y;
        return val;
}
double input(void)                         // 함수의 정의(10 형태)
{
        double val;
        scanf("%lf", &val);
        return val;
}
void output(double x)                      // 함수의정의(01 형태)
{
        printf("나눗셈 결과: %lf ₩n", x);
        return;
}
void information(void)                     // 함수의정의(00 형태)
{
        printf("--- 프로그램 시작 ---₩n");
        return;
}
```

9.4 변수의 종류와 범위

9.4 변수의 종류와 범위 (1/22)

- ▶ 지역 변수(Local Variable)
- ▶ 전역 변수(Global Variable)
- ▶ 정적 변수(Static Variable)
- ▶ 외부 변수(Extern Variable)
- ▶ 레지스터 변수(Register Variable)

9.4 변수의 종류와 범위 (2/22)

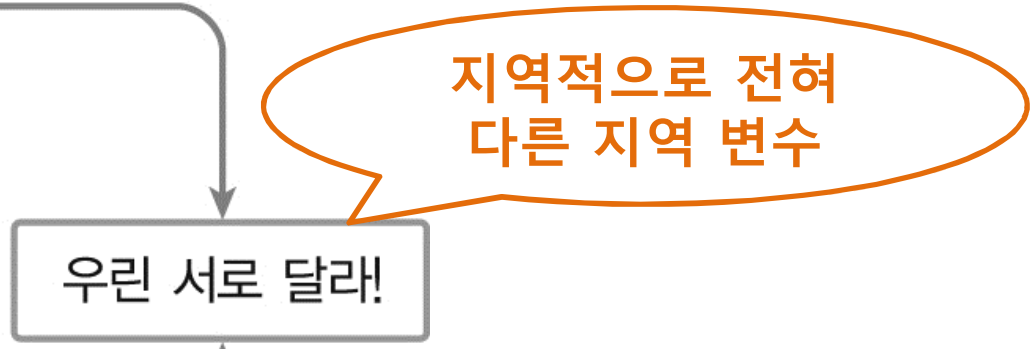
▶ 지역 변수(Local Variable)

✓ 사용 범위

- 함수 내부에서 사용
- 조건문 또는 반복문의 중괄호({ }) 내부에서 사용
- 함수의 매개 변수(Parameter) 즉, 함수의 입력 변수로 사용

9.4 변수의 종류와 범위 (3/22)

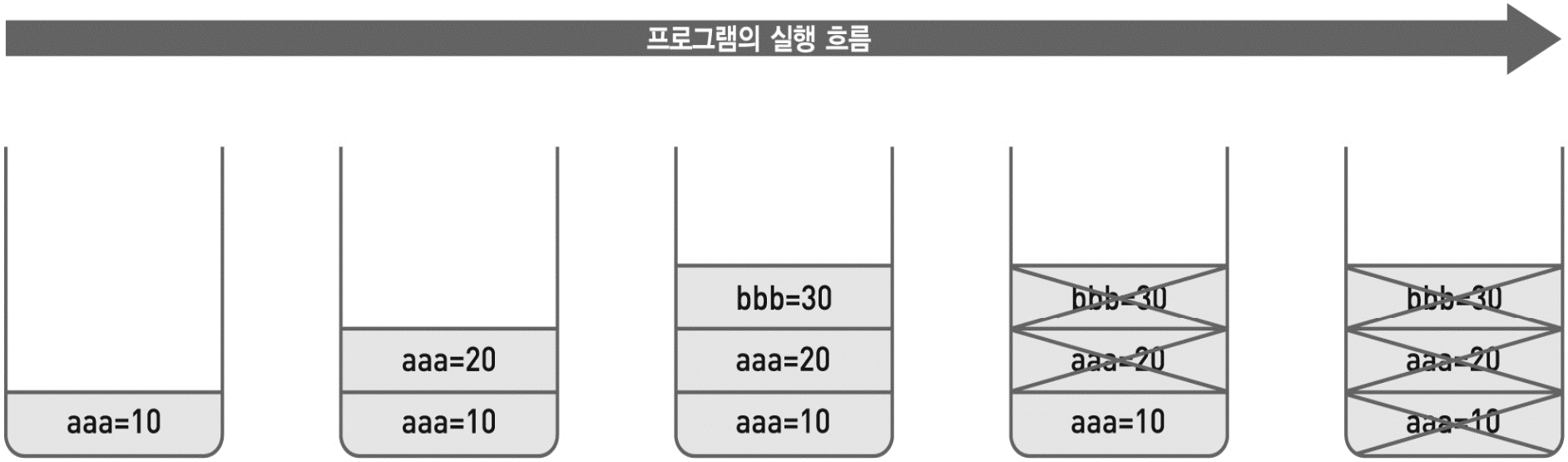
```
int sum(int x, int y)
{
    int result=0;
    result=x+y;
    return result;
}


int main(void)
{
    int result=10;
    result=sum(3, 4);
    printf("%d ₩n", result);
    return 0;
}
```



9.4 변수의 종류와 범위 (4/22)---[9-5.c 실습]

```
#include <stdio.h>
void func_A (void);

int main(void)
{
        int aaa=10;                          // main( ) 함수의 지역변수aaa
        printf("main( ) 함수의 aaa 값: %d₩n", aaa );

        func_A( );
        return 0;
}
void func_A(void)
{
        int aaa=20;                          // func_A( ) 함수의 지역변수 aaa
        int bbb=30;                          // func_A( ) 함수의 지역변수 bbb

        printf("func_A( ) 함수의 aaa 값: %d₩n", aaa );
        printf("func_A( ) 함수의 bbb 값: %d₩n", bbb );
        return ;
}
```

9.4 변수의 종류와 범위 (5/22)



6행에서 main( ) 함수의 지역 변수 aaa를 위해 메모리 공간 생성

15행에서 func_A( ) 함수의 지역 변수 aaa를 위해 메모리 공간 생성

16행에서 func_A( ) 함수의 지역 변수 bbb를 위해 메모리 공간 생성

20행에서 func_A( ) 함수 종료, 지역 변수 aaa와 bbb의 메모리 공간 소멸

10행에서 main( ) 함수 종료, 지역 변수 aaa의 메모리 공간 소멸

{ }(중괄호)지역을 빠져나가면 **메모리가 자동으로 소멸**

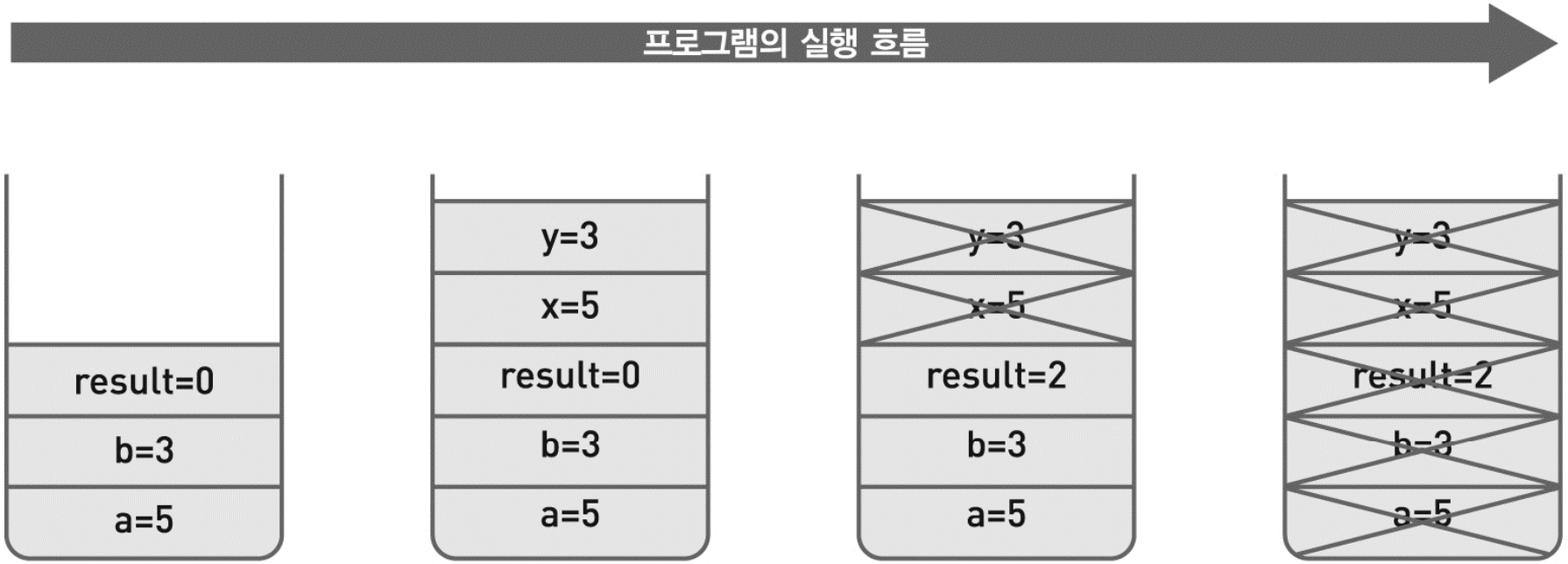
9.4 변수의 종류와 범위 (6/22)---[9-6.c 실습]

```
#include <stdio.h>
int main(void)
{
    int i=0;
    // int total=0;

    for(i=1; i<3; i++)
    {
        int total=0;     // 지역변수total 선언
        total=total+i;
    }

    if(total<10)         // 에러발생
    {
        printf("total 값은 %d입니다.₩n", total);
    }

    return 0;
}
```

9.4 변수의 종류와 범위 (7/22)---[9-7.c 실습]

```
#include <stdio.h>
int subtract(int x, int y);                    // 함수의 선언(11 형태)

int main(void)
{
        int a=5, b=3;
        int result=0;

        result=subtract(a, b);                 // 함수의 호출
        printf("뺄셈결과: %d ₩n", result);
        return 0;
}
int subtract(int x, int y)                     // 함수의 정의
{
        return x-y;
}
```

9.4 변수의 종류와 범위 (8/22)



6행에서 main( ) 함수의 지역 변수 a, b, result를 위해 메모리 공간 생성

13행에서 변수 a, b의 값을 매개 변수 x, y에 복사, x와 y의 메모리 공간 생성

15행에서 subtract( ) 함수의 결과 x-y를 main( ) 함수의 지역 변수 result에 반환하고, subtract( ) 함수가 종료되어 매개 변수 x, y의 메모리 공간 소멸

11행에서 main( ) 함수가 종료되어 지역 변수 a, b, result의 메모리 공간 소멸

9.4 변수의 종류와 범위 (9/22)

▶ 지역 변수의 특징 정리

- ✓초기화를 하지 않으면 쓰레기 값이 저장됨
- ✓**지역 변수의 메모리 생성 시점:** 중괄호 내에서 초기화할 때
- ✓**지역 변수의 메모리 소멸 시점:** 중괄호를 탈출할 때

9.4 변수의 종류와 범위 (10/22)

▶ 전역 변수(Global Variable)

✓ 사용 범위

- 중괄호({ }) 외부에서 사용

```
int z=0          ← 전역 변수 선언

int main(void)
{
        sum(1, 2);
        return 0;
}

void sum(int x, int y)   ← 지역 변수(매개 변수)
{
        z=x+y;
}
```

9.4 변수의 종류와 범위 (11/22)---[9-8.c 실습]

```
#include <stdio.h>
int num;                        // 전역변수선언, 초기화하지 않아도 0 설정
void grow(void);

int main(void)
{
        printf("함수 호출 전num : %d ₩n", num);    // 0 출력

        grow( );                        // 함수 호출
        printf("함수 호출 후 num : %d ₩n", num);

        return 0;
}

void grow(void)
{
        num=60;                 // 전역변수 num의 값 변경
}
```

9.4 변수의 종류와 범위 (12/22)

▶ 전역 변수의 특징

- ✓초기화를 하지 않아도 자동으로 0 설정
- ✓**전역 변수의 메모리 생성 시점:** 프로그램이 시작될 때
- ✓**전역 변수의 메모리 소멸 시점:** 프로그램이 종료될 때

9.4 변수의 종류와 범위 (13/22)

▶ 정적 변수(Static Variable)

```
static int num;
```

- ✓ 자료형 앞에 static 키워드를 붙임
- ✓ 프로그램이 종료되지 않는 한 메모리가 소멸되지 않음
- ✓ 초깃값을 지정하지 않아도 자동으로 0을 가짐
- ✓ 프로그램이 시작되면 초기화는 딱 한 번만 수행

9.4 변수의 종류와 범위 (14/22)---[9-9.c 실습]

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

        return 0;
}
void count(void)
{
        static int x=0;     // 정적 변수, 초기화를 한 번만 수행
        int y=0;            // 지역 변수, 초기화를 매 번 수행

        x=x+1;
        y=y+1;

        printf("x 값: %d, y 값: %d ₩n", x, y);
}
```

'정적 변수는 **중괄호가 있는 지역**에서
전역 변수의 기능이 필요할 때 사용한다.'

9.4 변수의 종류와 범위 (15/22)

▶ 정적 변수의 특징

- ✓초기화를 하지 않아도 자동으로 0 설정
- ✓초기화는 한 번만 수행
- ✓**정적 변수의 메모리 생성 시점:** 중괄호 내에서 초기화될 때
- ✓**정적 변수의 메모리 소멸 시점:** 프로그램이 종료될 때

9.4 변수의 종류와 범위 (16/22)

▶ 외부 변수

- ✓ 외부 파일에 선언된 변수를 참조하는 변수
- ✓ 자료형 앞에 **extern** 키워드를 사용
- ✓ 다른 파일(외부)에 있는 전역 변수를 참조

9.4 변수의 종류와 범위 (17/22)

test.c

```
int num1=5;       // 파일 test.c의 전역 변수
int num2=10;      // 파일 test.c의 전역 변수
int num3=20;      // 파일 test.c의 전역 변수

void add(void)
{
      num3=num1+num2;
}
```

9.4 변수의 종류와 범위 (18/22)

test1.c

```
#include <stdio.h>

int main(void)
{
    extern int num1;   // 외부 변수
    extern int num2;   // 외부 변수
    extern int num3;   // 외부 변수

    printf("num1의 값 : %d ₩n", num1);          // 5 출력
    printf("num2의 값 : %d ₩n", num2);          // 10 출력
    printf("num3의 값 : %d ₩n", num3);          // 20 출력
    printf("덧셈 결과 : %d ₩n", num1+num2+num3); // 35 출력
}
```

9.4 변수의 종류와 범위 (19/22)

▶ 특정 전역 변수를 외부에서 참조 못하게 하려면?

✓ **static** 키워드를 사용

test.c

```
int num1=5;                 // 전역 변수
int num2=10;                // 전역 변수
static int num3=20;         // 정적 전역 변수

void add(void)
{
      num3=num1+num2;
}
```

9.4 변수의 종류와 범위 (20/22)

▶ 레지스터 변수(Register Variable)

- ✓CPU 내부의 레지스터에 변수를 할당하는 변수
- ✓**처리속도가 빠름**

9.4 변수의 종류와 범위 (21/22)---[9-10.c 실습]

```
#include <stdio.h>
#include <time.h>                    // 연산 속도 측정을 위해 clock( ) 함수 사용
#define MAX 1000000                  // 백만을 상수화

int main(void)
{
        register int i;              // int i;
        clock_t startTime, endTime, result;

        startTime=clock( );          // startTime : 측정 시작
         for (i=0; i<=MAX; i++)
         {
             printf("%d₩n", i);
         }
         endTime=clock( );           // endTime : 측정 완료

        result=endTime-startTime;    // 연산 속도
        printf("레지스터 변수 속도: %lf초 ₩n", (double)result/1000);
        return 0;
}
```

9.4 변수의 종류와 범위 (22/22)

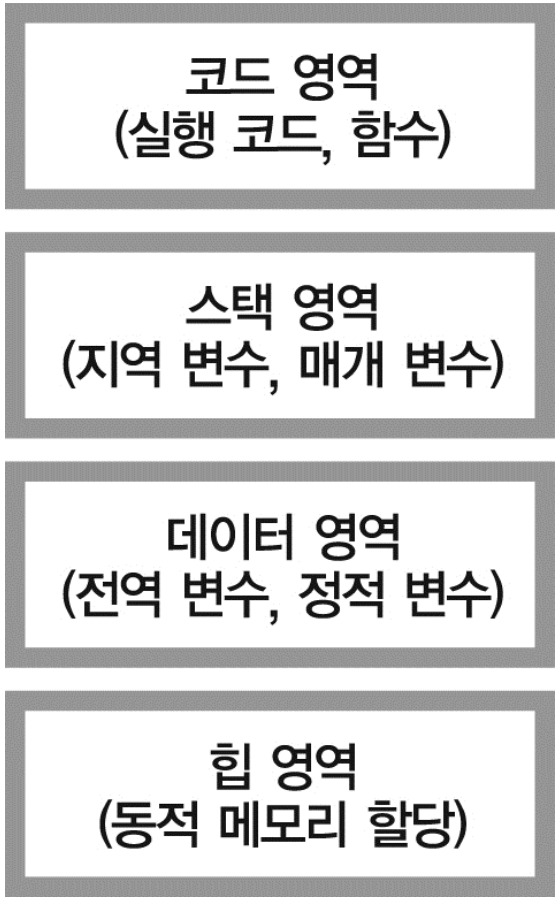
▶프로세스의 메모리 구조

✓ **코드 영역** : 프로그램의 **실행 코드** 또는 **함수**들이 저장되는 영역

✓ **스택 영역** : **매개 변수 및 중괄호(블록) 내부에 정의된 변수**들이 저장되는 영역

✓ **데이터 영역** : **전역 변수**와 **정적 변수**들이 저장되는 영역

✓ **힙 영역** : **동적으로 메모리 할당하는 변수**들이 저장되는 영역



9.5 재귀 함수

9.5 재귀 함수 (1/5)

▶ 재귀 함수(Recursive Function)

- ✓ 함수 내에서 **자기 자신을 호출**하는 함수
- ✓ 재귀 호출(Recursive Call) : 자기 자신을 호출하는 행위

▶ 재귀 호출의 문제점

- ✓ 시간과 메모리 공간의 효율이 저하
  - ➔ 개발에 신중해야 함

9.5 재귀 함수 (2/5)---[9-11.c 실습]

```
#include <stdio.h>
void self_service(void);                    // 함수의 선언(00 형태)

int main(void)
{
        self_service( );                    // 함수의 호출
        return 0;
}

void self_service(void)                     // 함수의 정의
{
        printf("셀프서비스₩n");
        self_service( );
}
```

9.5 재귀 함수 (3/5)---[9-12.c 실습]

```
#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( );
}
```

9.5 재귀 함수 (4/5)---[9-13.c 실습]

```
#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
}
```

9.5 재귀 함수 (5/5)---[9-14.c 실습]

```
#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)                    // 함수의 정의
{
        if (n<=1)
            return 1;
        else
            return n * factorial(n-1);
}
```

공부한 내용 떠올리기

► 함수 : 특별한 일을 수행하는 코드의 집합

► 함수의 다양한 입출력 형태 4가지 (11, 10, 01, 00 형태)

► 함수의 적용방법 2가지
- ✓ 함수의 정의, 함수의 호출
- ✓ 함수의 선언, 함수의 호출, 함수의 정의

► 변수의 종류와 범위
- ✓ 지역 변수
- ✓ 전역 변수
- ✓ 정적 변수
- ✓ 외부 변수
- ✓ 레지스터 변수

► 재귀함수란 무엇인가?

실패와 교훈(출처: 사랑과 지혜의 탈무드)