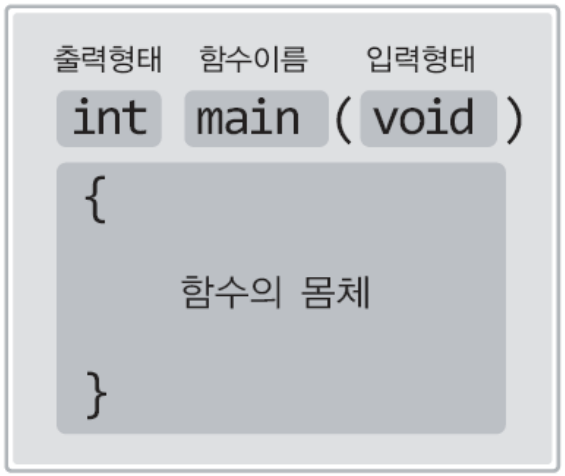
함수(1)

- 함수의 정의와 선언, 재귀 함수 -

성공회대학교 IT융합자율학부
소프트웨어공학전공
홍 성 준

함수를 정의하고 선언하기

◎ C언어의 함수 (function)



함수를 정의하고 선언하기

◎ 함수를 만드는 이유

- 프로그램 구현은 복잡한 문제를 해결하는 것
- 프로그램 구현에 필요한 기능을 분석하고, 이를 바탕으로 작은 단위의 함수를 디자인하여 구현함
- 버그를 수정하거나 프로그램 요구사항의 변경으로 인한 소스 코드 수정 범위를 축소하고 제한할 수 있음

함수를 정의하고 선언하기

◎ 함수의 입력과 출력

- printf() 함수의 반환 값?

```
int main(void)
{
    int num1, num2;
    num1=printf("12345\n");
    num2=printf("I love my home\n");
    printf("%d %d \n", num1, num2);
    return 0;
}
```

```
12345
I love my home
6 15
```

- printf 함수도 출력된 문자열의 길이를 반환하지만, 반환한 값이 필요 없어 따로 저장하지 않음

함수를 정의하고 선언하기

◎ 전달인자 유무와 반환값의 유무에 따른 함수의 형태

유형 1: 전달인자 있고, 반환 값 있다! 전달인자(○), 반환 값(○)

유형 2: 전달인자 있고, 반환 값 없다! 전달인자(○), 반환 값(×)

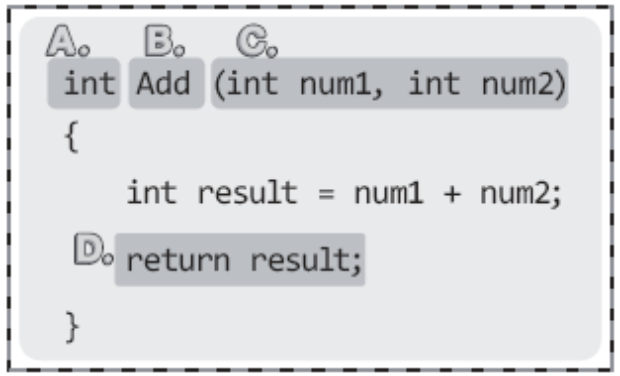
유형 3: 전달인자 없고, 반환 값 있다! 전달인자(×), 반환 값(○)

유형 4: 전달인자 없고, 반환 값 없다! 전달인자(×), 반환 값(×)

함수를 정의하고 선언하기

◎ 전달인자와 반환 값이 모두 있는 경우

- 가장 일반적인 함수의 형태



A. 반환형
B. 함수의 이름
C. 매개변수
D. 값의 반환

- 전달인자: int형 정수값 2개
- 반환값: int형 정수값 (덧셈결과)

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

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

```
덧셈결과1: 7
덧셈결과2: 13
```

함수를 정의하고 선언하기

◎ 전달인자나 반환 값이 존재하지 않는 경우

```
void ShowAddResult(int num)   // 인자전달 (O), 반환 값 (X)
{
    printf("덧셈결과 출력: %d \n", num);
}
```

```
int ReadNum(void)   // 인자전달 (X), 반환 값 (O)
{
    int num;
    scanf("%d", &num);
    return num;
}
```

```
void HowToUseThisProg(void)   // 인자전달 (X), 반환 값 (X)
{
    printf("두 개의 정수를 입력하시면 덧셈결과가 출력됩니다. \n");
    printf("자! 그럼 두 개의 정수를 입력하세요. \n");
}
```

함수를 정의하고 선언하기

◎ SmartAddFunc.c

```
int Add(int num1, int num2)    // 인자전달 (O), 반환 값 (O)
{
    return num1+num2;
}

void ShowAddResult(int num)    // 인자전달 (O), 반환 값 (X)
{
    printf("덧셈결과 출력: %d \n", num);
}

int ReadNum(void)    // 인자전달 (X), 반환 값 (O)
{
    int num;
    scanf("%d", &num);
    return num;
}

void HowToUseThisProg(void)    // 인자전달 (X), 반환 값 (X)
{
    printf("두 개의 정수를 입력하시면 덧셈결과가 출력됩니다. \n");
    printf("자! 그럼 두 개의 정수를 입력하세요. \n");
}
```

```
int main(void)
{
    int result, num1, num2;
    HowToUseThisProg();
    num1=ReadNum();
    num2=ReadNum();
    result = Add(num1, num2);
    ShowAddResult(result);
    return 0;
}
```

```
두 개의 정수를 입력하시면 덧셈결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
12 24
덧셈결과 출력: 36
```

함수를 정의하고 선언하기

◎ 값을 반환하지 않은 return

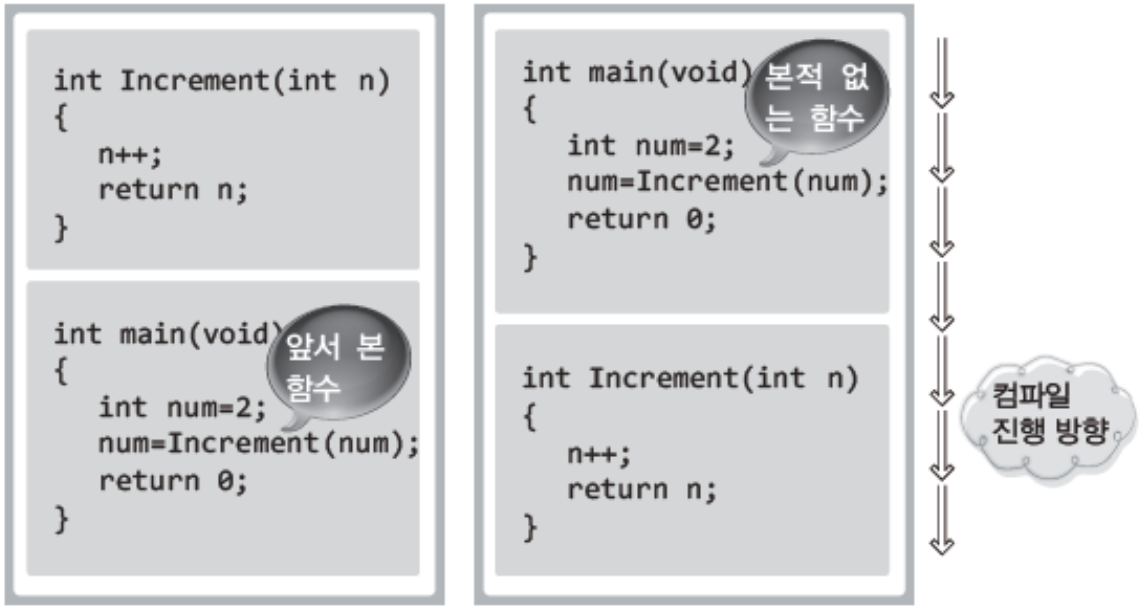
- return 문에는 '값의 반환'과 '함수 탈출' 기능이 있음
- 값을 반환하지 않는 형태로 return 문을 구성하여 반환 값이 없는 채로 함수를 빠져나갈 수 있음

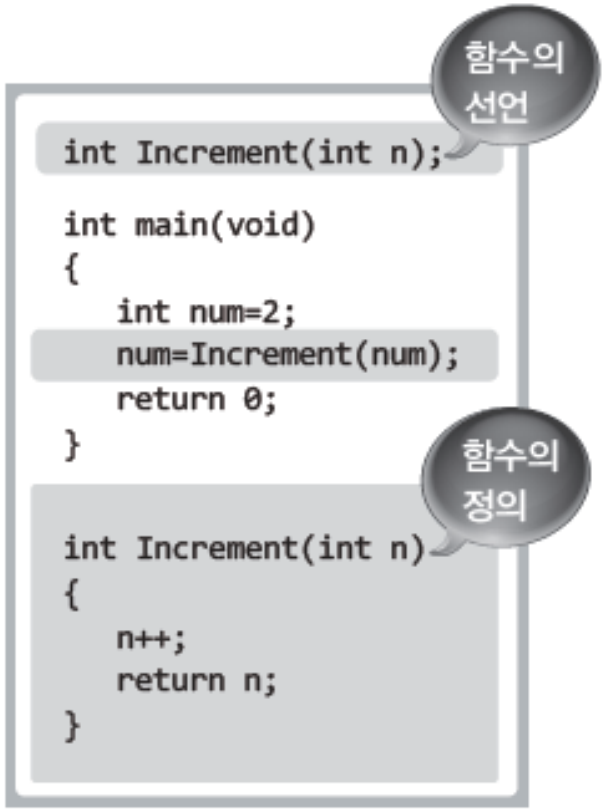
```
void NoReturnType(int num)
{
    if(num<0)
        return;    // 값을 반환하지 않는 return문!
    . . . .
}
```

함수를 정의하고 선언하기

◎ 함수의 정의와 그에 따른 원형의 선언

- 소스 코드의 컴파일은 위에서부터 아래로 진행되기 때문에 함수의 배치 순서가 중요함
- 실제 컴파일 되지 않은 함수는 호출이 불가능함





- 소스 코드 앞에서 사용할 함수 정보(함수의 선언)를 컴파일러에 제공하여, 함수 호출 구문이 컴파일 되도록 함

```
int Increment(int n);   // 함수의 선언
int Increment(int);     // 위와 동일한 함수선언, 매개변수 이름 생략 가능
```

함수를 정의하고 선언하기

◎ AbsoCompare.c

```
int AbsoCompare(int num1, int num2);   // 절댓값이 큰 정수 반환
int GetAbsoValue(int num);     // 전달인자의 절댓값을 반환

int main(void)
{
    int num1, num2;
    printf("두 개의 정수 입력: ");
    scanf("%d %d", &num1, &num2);
    printf("%d와 %d중 절댓값이 큰 정수: %d \n",
        num1, num2, AbsoCompare(num1, num2));
    return 0;
}
int AbsoCompare(int num1, int num2)
{
    if(GetAbsoValue(num1) > GetAbsoValue(num2))
        return num1;
    else
        return num2;
}
```

```
int GetAbsoValue(int num)
{
    if(num<0)
        return num * (-1);
    else
        return num;
}
```

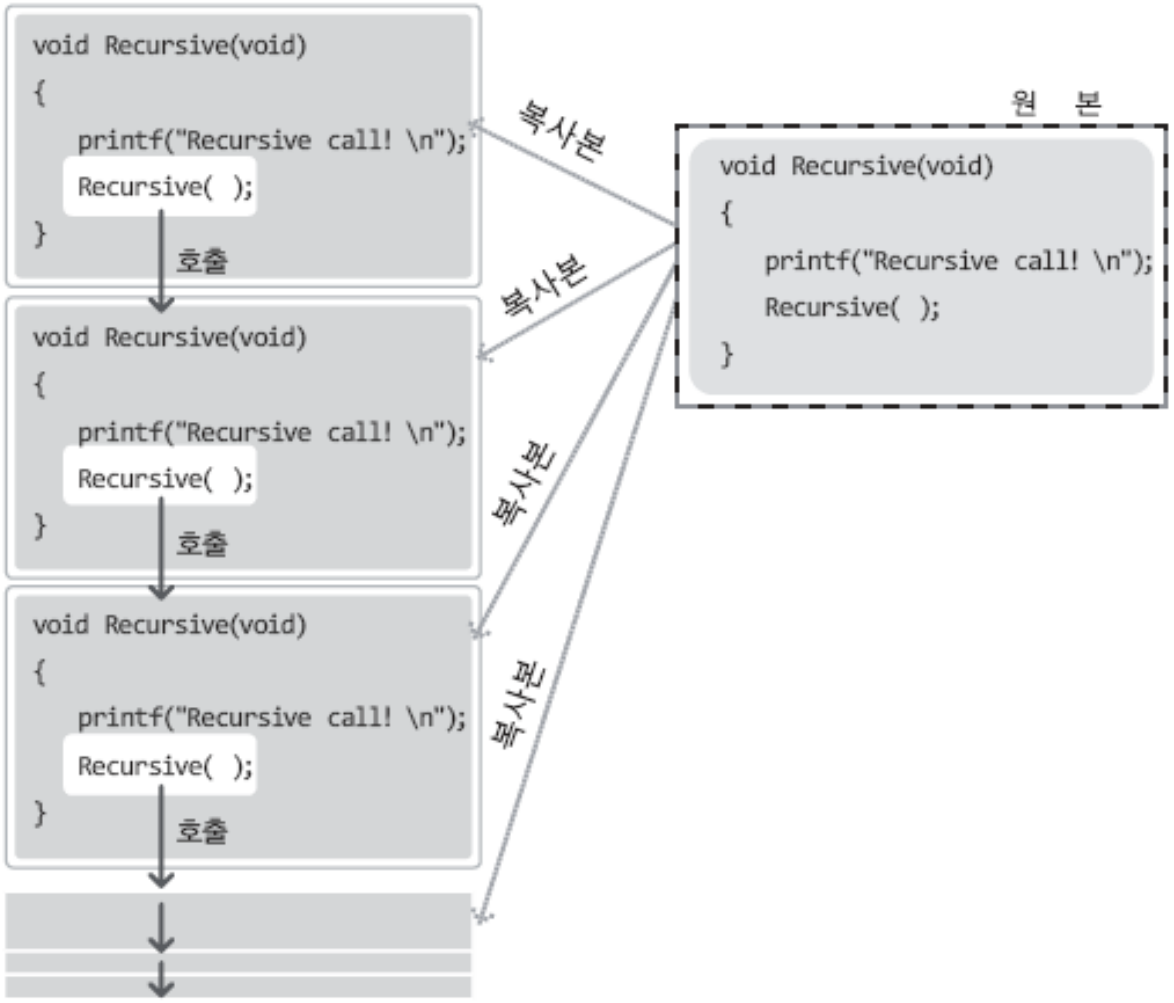
```
두 개의 정수 입력: 5 -9
5와 -9중 절댓값이 큰 정수: -9
```

재귀 함수

◎ 재귀 함수 (recursive function)

- 함수 내에서 자기 자신을 다시 호출하는 형태의 함수
- 자료 구조나 알고리즘 문제를 단순화 하는데 주로 사용

```
void Recursive(void)
{
    printf("Recursive call! \n");
    Recursive();   // 나! 자신을 재 호출한다.
}
```

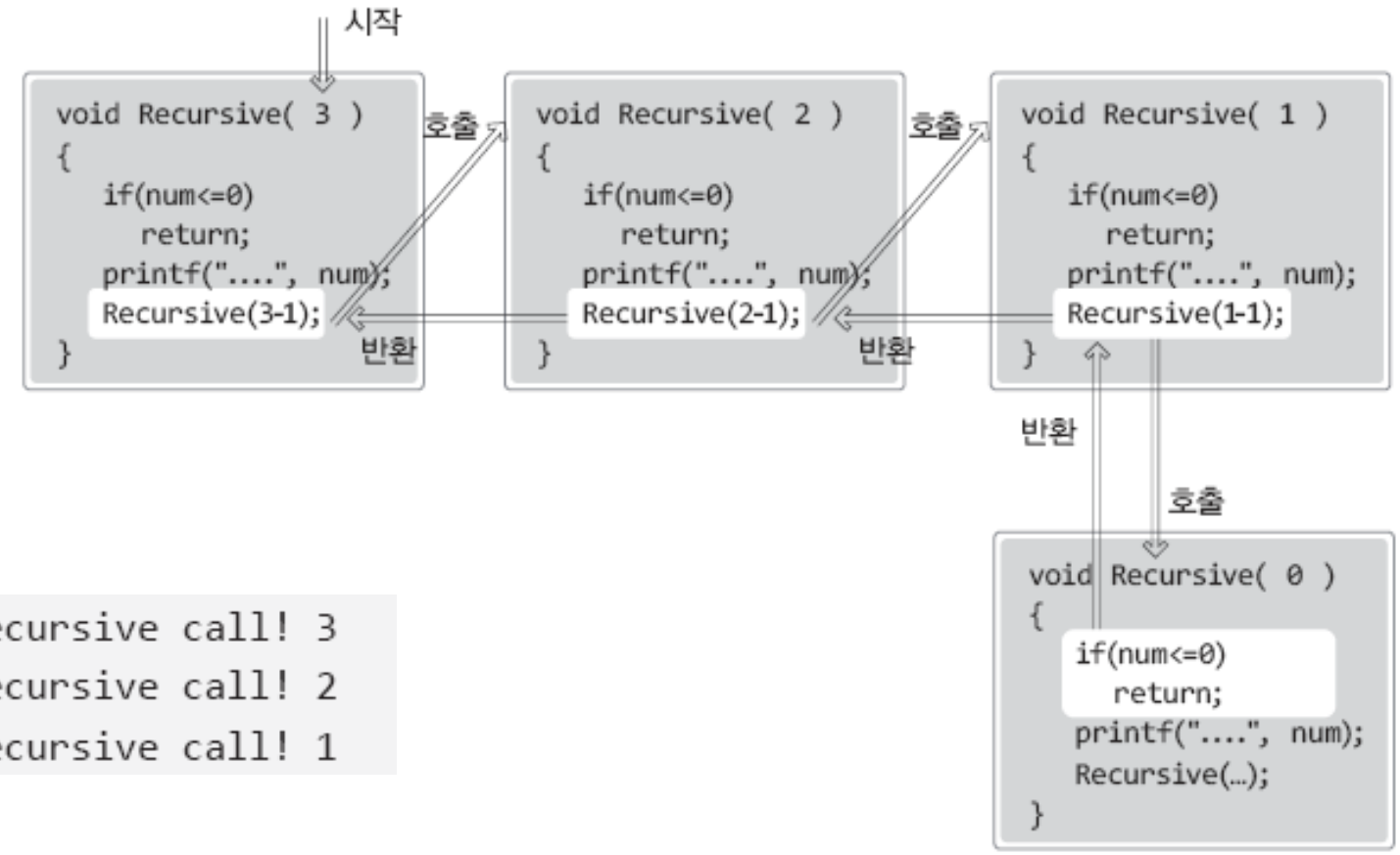


재귀 함수

◎ 재귀 함수 (recursive function)

```
void Recursive(int num)
{
    if(num<=0)      // 재귀의 탈출조건
        return; //재귀의 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
```

재귀 함수

◎ 계승 구하기 (factorial)

- $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$
- $n! = n \times (n-1)!,\ 0! = 1$
- $f(n) = n \times f(n-1),\ f(0) = 1$

$$f(n) = \begin{cases} n \times f(n-1) & \cdots\cdots n \geq 1 \\ 1 & \cdots\cdots n = 0 \end{cases}$$

```
if(n>=1)
    return n * Factorial(n-1);
if(n==0)
    return 1;
```

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

재귀 함수

◎ 계승 구하기 (factorial)

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

int main(void)
{
    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
```