Kumoh National Institute of Technology

C Programming

Ch.4 C 언어의 핵심! 함수!

컴퓨터공학과

C프로그래밍

Contents

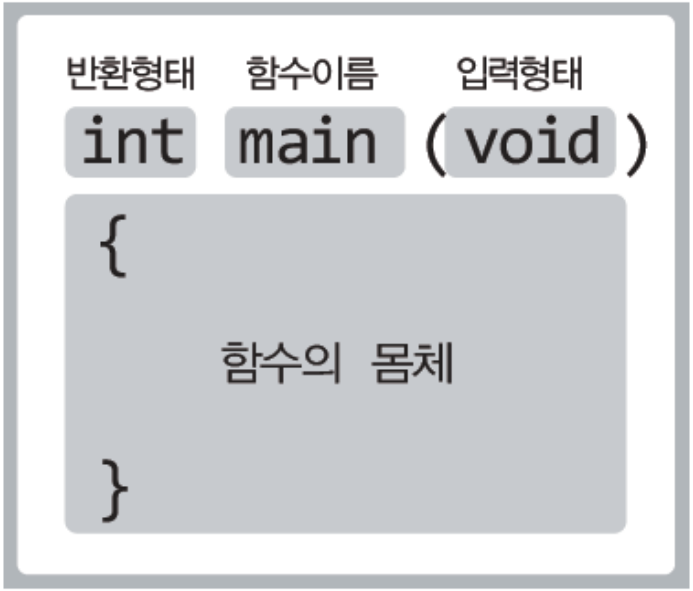
1. 함수를 정의하고 선언하기
2. 변수의 존재 기간과 접근 범위

   : 지역 변수
3. 변수의 존재 기간과 접근 범위

   : 전역 변수, static 변수, register 변수
4. 재귀 함수에 대한 이해

함수를 만드는 이유

- 함수를 만드는 이유
  - 문제 해결 용이
    - 문제 해결 방법 : Divide and Conquer!
    - 하나의 문제를 작은 단위로 나누어 해결 (하향식 설계)
    - 각 단위 별로 함수를 만들어 사용
  - 유지 보수 및 확장 용이
- main 함수 다시 보기



가능하다면 하나의 함수는 하나의 일만 담당하도록 설계
→ "하나의 일" : 주관적일 수도 있음

함수의 입력과 출력

- printf 함수의 입력과 반환

```
int main(void)
{
    int num1, num2, num3;

    num1 = printf("12345\n");
    num2 = printf("I love my home\n");
    num3 = printf("%d\n", 12345);

    printf("%d %d %d \n", num1, num2, num3);
}
```

printf 함수도 값을 반환함
- 출력된 문자열의 길이(특수문자 포함)
- 사용하지 않아도 됨

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

입력이 없거나 출력이 없는 함수도 만들 수 있음!

함수의 구분

- 전달인자의 유무와 반환값의 유무에 따른 함수의 구분

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(×)

전달인자와 반환값이 모두 있는 경우 (1)

- 함수의 이름 : Add
- 전달인자 : int 형 값 2개(를 저장할 변수들)
- 반환형 : int 형
- 함수의 내용 : 전달인자로 넘어온 2개의 정수를 더해 반환한다.

함수 정의
(함수 그 자체)

```
A.  B.  C.
int Add (int num1, int num2)
{
    int result = num1 + num2;
D.  return result;
}
```

A. 반환형
B. 함수의 이름
C. 매개변수
D. 값의 반환

전달인자와 반환값이 모두 있는 경우 (2)

- 함수를 만들었다면 사용해야지! → 함수 호출

```
int Add(int num1, int num2)
{
    int num3 = num1 + num2;
    return num3;
}

int main(void)
{
    int result;

    result = Add(3, 4);
    printf("덧셈결과1: %d \n", result);
    result = Add(5, 8);
    printf("덧셈결과2: %d \n", result);
}
```

```
덧셈결과1: 7
덧셈결과2: 13
```

// 다음 한 줄과 동일
return num1 + num2;

함수 호출이 완료되면 호출한 위치로 값이 반환되고 다음 줄부터 계속해서 실행됨

용어 정리
- 실 매개변수 (= 전달인자)
  · actual parameter (argument)
- 형식 매개변수 (= 매개변수)
  · formal parameter (parameter)

전달인자가 존재하지 않는 경우

```
int ReadNum(void)
{
    int num;

    printf("정수 1개 입력 : ");
    scanf("%d", &num);
    return num;
}

int main(void)
{
    int num1 = ReadNum();
    int num2 = ReadNum();

    printf("%d %d \n", num1, num2);
}
```

전달인자가 없을 경우 : void

```
정수 1개 입력 : 100
정수 1개 입력 : 200
100 200
```

반환값이 존재하지 않는 경우

반환값이 없을 경우 반환 타입은 :
void

return 문 생략 가능. 그러나
강제 탈출 시 사용 가능
→ 11페이지 참고

```
void ShowAddResult(int num1, int num2)
{
    printf("합계 : %d \n", num1 + num2);
}

int main(void)
{
    ShowAddResult(3, 4);
    ShowAddResult(100, 200);
}
```

```
합계 : 7
합계 : 300
```

전달인자와 반환값 모두 존재하지 않는 경우

```
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)
{
    HowToUseThisProg();
}
```

```
두 개의 정수를 입력하시면 덧셈결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
```

4가지 함수 유형을 조합한 예

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

void ShowAddResult(int num)      // 인자전
{
    printf("덧셈결과 출력: %d \n", num);
}

int ReadNum(void)      // 인자전달 (X), 반
{
    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);
}
```

```
두 개의 정수를 입력하시면 덧셈결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
100 200
덧셈결과 출력: 300
```

값을 반환하지 않는 return 문

- 반환 타입이 void인 함수 (값을 반환하지 않는 경우)
  - return 문 없어도 됨 : 마지막까지 실행하고 빠져나감
  - 반환값 명시 없이 return 문 사용 가능 : 함수 탈출

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

return 문의 두 가지 기능
- 값의 반환
- 함수 탈출

함수의 정의와 그에 따른 함수의 선언 (1)

- 함수 선언의 필요성
  - 컴파일러의 프로그램 해석 방향 : 위 → 아래
  - 함수를 호출하기 전에 함수에 대한 정체가 나와 있어야 함
    - 함수 정의 → Ok! 또는
    - 함수의 형태(함수명, 전달인자 타입, 반환 타입) : 함수 선언

```
int Increment(int n)
{
   n++;
   return n;
}

int main(void)
{
   int num=2;
   num=Increment(num);
   return 0;
}
```

앞서 본 함수

```
int main(void)
{
   int num=2;
   num=Increment(num);
   return 0;
}

int Increment(int n)
{
   n++;
   return n;
}
```

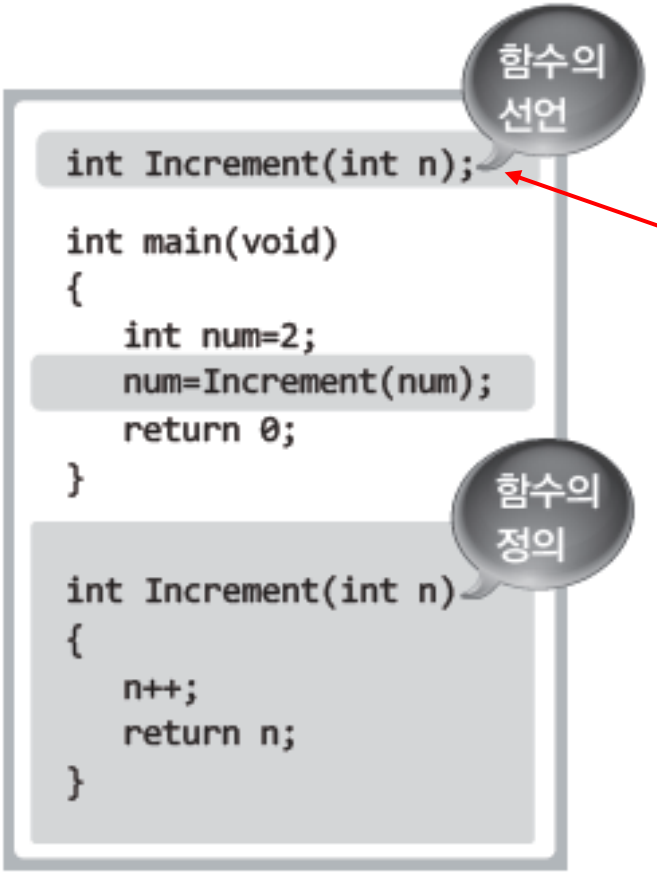
본적 없는 함수

에러 발생. 그런데 main 함수가 먼저 나오는 것이 프로그램 해석에 편함

컴파일 진행 방향

함수의 정의와 그에 따른 함수의 선언 (2)

- 함수 선언의 의미
  - 이후(또는 다른 파일)에 정의될 함수에 대한 정보 제공



함수 선언 = 함수 원형 = 함수 프로토타입
함수 원형에는 전달 인자의 이름은 안 붙여도 됨 생략가능

```
int Increment(int);
int Add(int, int);
```

함수 정의는 함수 선언의 기능(정보 제공)을 포함하고 있음
따라서 기존 함수 선언을 “순수 선언”이라고 부르기도 함

함수의 작성 및 사용 예 (1)

```
int NumberCompare(int num1, int num2);

int main(void)
{
    printf("3과 4중에서 큰 수는 %d 이다. \n", NumberCompare(3, 4));
    printf("7과 2중에서 큰 수는 %d 이다. \n", NumberCompare(7, 2));
}

int NumberCompare(int num1, int num2)
{
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

```
3과 4중에서 큰 수는 4 이다.
7과 2중에서 큰 수는 7 이다.
```

함수의 작성 및 사용 예 (2)

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

int main(void)
{
    int num1, num2;
    printf("두 개의 정수 입력: ");
    scanf("%d %d", &num1, &num2);
    printf("%d와 %d중 절대값이 큰 정수: %d \n",
            num1, num2, AbsoCompare(num1, num2));
}

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;
}
```

소스 코드를 읽는 두 가지 방법

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

변수의 존재 기간과 접근 범위

- 접근 범위에 따른 변수의 분류 (scope)
  - 지역 변수 (local) : 지역 내에서만 사용할 수 있는 변수
  - 전역 변수 (global) : 프로그램 전체에서 사용 가능한 변수
  - static 전역 변수 : 해당 파일 내에서 (전역 변수처럼) 사용 가능한 변수
- 저장 방식(≒존재 기간)에 따른 변수의 분류 (storage class)
  - 자동 변수 (auto) : 해당 지역 시작 ~ 지역 종료
    - 지역 변수 디폴트
  - 정적 변수 (static) : 프로그램 시작 ~ 프로그램 종료
    - 지역 변수, 전역 변수 모두 가능
  - 레지스터 변수 (register) : 해당 지역 시작 ~ 지역 종료
    - register 메모리에 할당 요청
    - 지역 변수에 사용 가능

```
int main(void)
{
    int num1;    // = auto int num;
    static int num2;
    register int num3;
}
```

지역 변수 : 함수 내에만 존재 및 접근 가능

- 지역 변수 : 지역(함수 등) 내에 선언되는 변수
  - 함수 내에서만 접근 가능
  - 해당 지역 진입 시 생성되며 그 지역을 벗어날 때 소멸

```
int SimpleFuncOne(void)
{
    int num = 10;    // 이후부터 SimpleFuncOne의 num 유효
    num++;
    printf("SimpleFuncOne num: %d \n", num);
    return 0;        // SimpleFuncOne의 num 유효한 마지막 문장
}

int SimpleFuncTwo(void)
{
    int num1 = 20;  // 이후부터 num1 유효
    int num2 = 30;  // 이후부터 num2 유효
    num1++, num2--;
    printf("num1 & num2: %d %d \n", num1, num2);
    return 0;    // num1, num2 유효한 마지막 문장
}
```

```
SimpleFuncOne num: 11
num1 & num2: 21 29
main num: 17
```

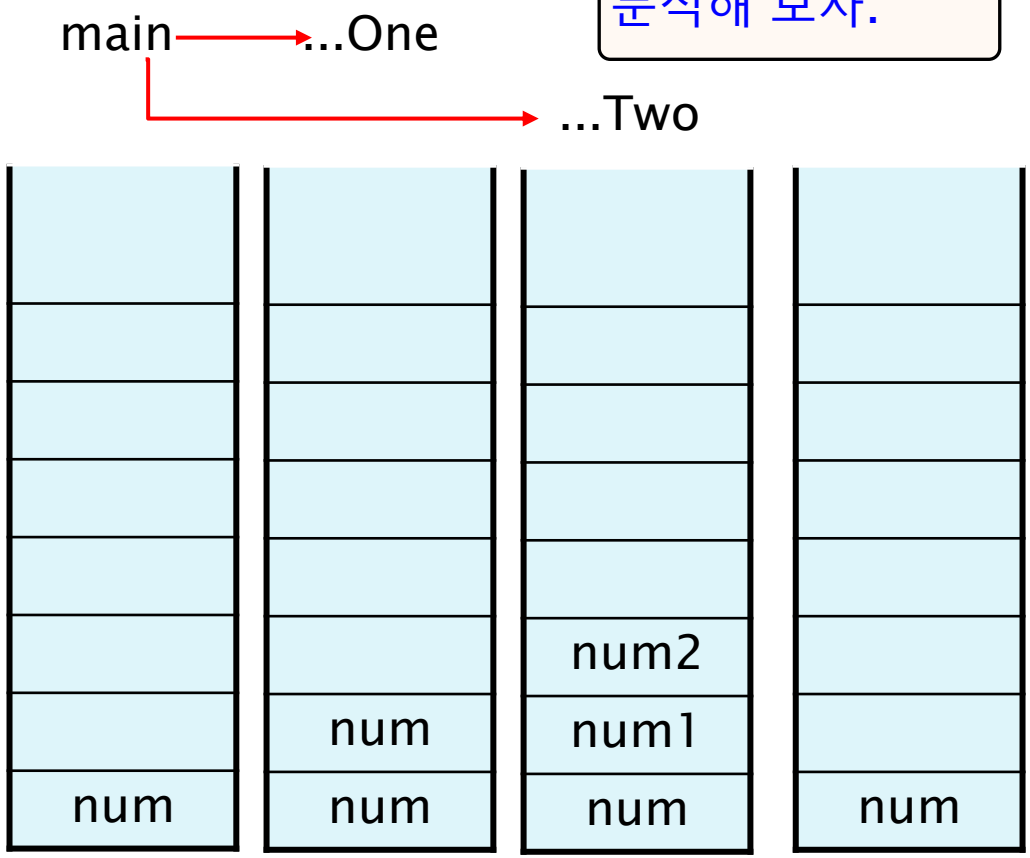
```
int main(void)
{
    int num = 17;
    SimpleFuncOne();
    SimpleFuncTwo();
    printf("main num: %d \n", num);
    return 0;
}
```

메모리 공간의 할당과 소멸 관찰하기

- 지역 변수는 스택이라는 메모리 영역에 생김
  - 스택(stack) : FILO(first-in last-out) 구조

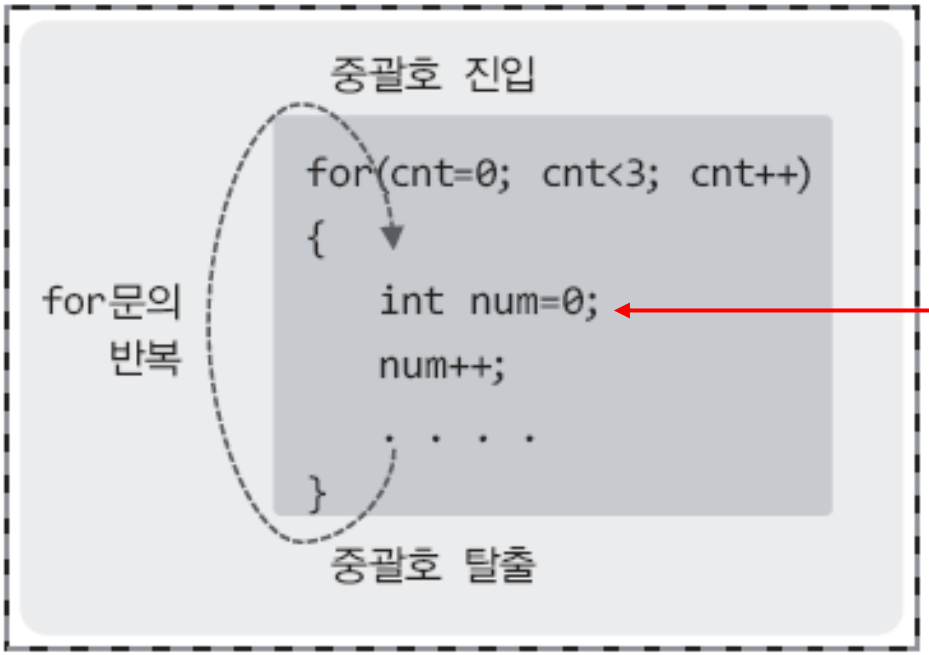
값의 변화도 함께 분석해 보자.

```
int SimpleFuncOne(void)
{
    int num = 10;
    num++;
    printf("SimpleFuncOne num: %d \n", num);
    return 0;
}

int SimpleFuncTwo(void)
{
    int num1 = 20;
    int num2 = 30;
    num1++, num2--;
    printf("num1 & num2: %d %d \n", num1, num2);
    return 0;
}

int main(void)
{
    int num = 17;
    SimpleFuncOne();
    SimpleFuncTwo();
    printf("main num: %d \n", num);
    return 0;
}
```



다양한 형태의 지역 변수

- while, for, if 문과 같은 제어문에 의해 형성되는 지역 내에서 선언되는 변수 → 해당 지역 내에서만 사용 가능



for 문을 벗어나면 소멸됨

변수 i도 for 지역 내의 지역 변수임

```
for (int i = 0; i < 10; i++)
    printf("hahaha");
```

지역 변수의 사용 예 (1)

```
int main(void)
{
    int cnt;
    for (cnt = 0; cnt < 3; cnt++)
    {
        int num = 0;
        num++;
        printf("%d번째 반복, 지역변수 num은 %d. \n", cnt + 1, num);
    }

    if (cnt == 3)
    {
        int num = 7;
        num++;
        printf("if문 내에 존재하는 지역변수 num은 %d. \n", num);
    }
}
```

```
1번째 반복, 지역변수 num은 1.
2번째 반복, 지역변수 num은 1.
3번째 반복, 지역변수 num은 1.
if문 내에 존재하는 지역변수 num은 8.
```

지역 변수의 사용 예 (2)

```
int main(void)
{
    int num = 1;

    if (num == 1)
    {
        int num = 7;      // 이 행을 주석처리 하고 실행결과 확인하자!
        num += 10;
        printf("if문 내 지역변수 num: %d \n", num);
    }

    printf("main 함수 내 지역변수 num: %d \n", num);
}
```

지역 변수는 외부에 선언된 동일한 이름의 변수를 가림

```
if문 내 지역변수 num: 17
main 함수 내 지역변수 num: 1
```

주석 처리 후의 실행 결과는?

지역 변수의 일종인 매개변수

- 매개 변수도 지역 변수

```
int Sum(int num1, int num2)
{
    int num3 = num1 + num2;

    return num3;
}

int main(void)
{
    int x = 3, y = 4;
    int z = Sum(x, y);

    printf("합계 : %d \n", z);
}
```

num1, num2도 지역 변수!

```
합계 : 7
```

(참고) 단순 블록도 지역이다

- 반복문 또는 조건문 없이 블록({ . . . })만 올 수 있음

```
int main(void)
{
    int a = 3;
    int b = 4;

    {
        int b = 100;
        a++;
        b++;

        printf("a = %d, b = %d\n", a, b);
    }

    printf("a = %d, b = %d\n", a, b);
}
```

이것도 하나의 지역

```
a = 4, b = 101
a = 4, b = 4
```

전역 변수의 이해와 선언 방법

- 전역 변수 : 함수 외부에 선언
  - 프로그램 어디서나 접근이 가능한 변수
  - 프로그램 시작부터 종료될 때까지 메모리에 할당되어 존재함

```
void Add(int val);
int num;     // 전역변수는 기본 0으로 초기화됨

int main(void)
{
    printf("num: %d \n", num);
    Add(3);
    printf("num: %d \n", num);
    num++;
    printf("num: %d \n", num);
}

void Add(int val)
{
    num += val;
}
```

전역 변수 선언 시 초기화하지 않으면 0으로 초기화됨
- cf) 지역 변수 : 쓰레기값으로 초기화됨

```
num: 0
num: 3
num: 4
```

전역변수와 동일한 이름의 지역 변수가 선언되면?

- 지역 변수의 이름이 전역 변수의 이름을 가림

```
int Add(int val);
int num = 1;

int main(void)
{
    int num = 5;
    printf("num: %d \n", Add(3));
    printf("num: %d \n", num + 9);
}

int Add(int val)
{
    int num = 9;
    num += val;
    return num;
}
```

```
num: 12
num: 14
```

(참고) 전역 변수도 선언 이후로 사용 가능

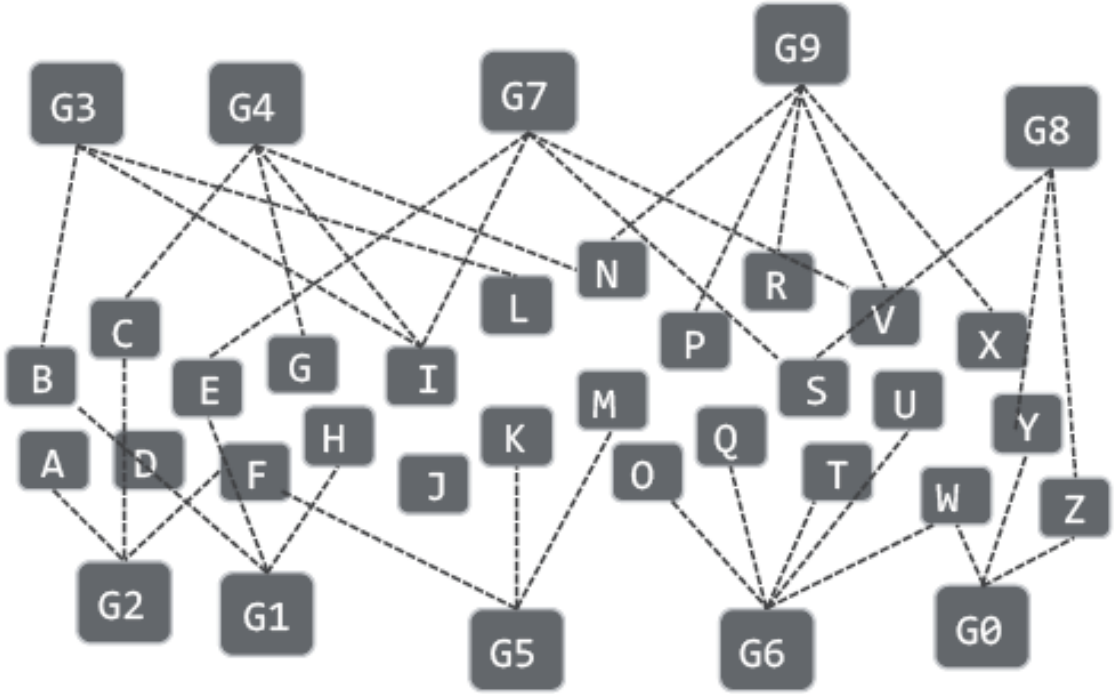
```
void Add(int val);

int main(void)
{
    printf("num: %d \n", num);  // num 사용 불가
    Add(3);
    printf("num: %d \n", num);
    num++;
    printf("num: %d \n", num);
}

int num;    // 여기에 선언한다면?

void Add(int val)
{
    num += val;
}
```

* extern 선언 (순수 선언)
  extern int num;
  - 어딘가에 int num;이 있다. 바로 그것이다!
  - 함수 외부/내부 모두 선언 가능
  - 함수의 경우 함수 프로토타입과 기능 유사

이 예에서는 int num;을 중간에 선언할 이유가 없으며, extern 선언을 할 이유가 없음
→ 다른 파일에 선언한 전역 변수를 이 파일에서 사용하려면?
→ extern 선언 필요 (교재 13주차)

전역 변수! 많이 써도 되는가?

- 많이 쓰면 좋지 않다.
  - 전역 변수 변경 → 프로그램 전체적으로 영향 미침
  - 전역 변수에 의존하는 코드 자제



G0~G9의 전역변수와 함수와의 접근관계의 예시

static 지역 변수 : 지역 변수에 static 추가

- static 지역 변수
  - 지역 변수와 같이 해당 지역 내에서만 사용 가능
  - 프로그램 시작 시 생성, 기본 초기값 0(초기값이 주어져 있다면 해당 값으로 초기화) ~ 프로그램 종료 시 소멸

```
void SimpleFunc(void)
{
    static int num1 = 0;
    int num2 = 0;

    num1++, num2++;
    printf("static: %d, local: %d \n", num1, num2);
}

int main(void)
{
    int i;
    for (i = 0; i < 3; i++)
        SimpleFunc();
}
```

static 변수의 초기화는 변수 생성 및 초기화 시에만 실행됨
→ 함수 호출 시 다시 0으로 초기화하지 않음

```
static: 1, local: 1
static: 2, local: 1
static: 3, local: 1
```

static 지역 변수는 자주 써도 되나요?

- static 지역 변수가 유용한 때
  - 함수를 빠져나가도 계속해서 그 값이 유지될 필요가 있다
    - 전역 변수
    - static 지역 변수
  - 특정 함수하고만 관련 있다.
    - 지역 변수
    - static 지역 변수
- static 지역 변수의 활용 예
  - Sum 이란 함수가 몇 번 호출되고 있나?
  - Minus 란 함수는 몇 번 호출되고 있나?
  - 각각의 함수 내에 static int call_count = 0; 선언!

(참고) static 전역 변수

- 해당 파일 내에서만 사용 가능한 전역 변수

```
static int result = 0;

void Sum(int num)
{
    result += num;
}

int main(void)
{
    for (int i = 1; i <= 10; i++)
        Sum(i);
    printf("합계 : %d \n", result);
}
```

다른 파일에서는 이 변수를 공유할 수 없음
→ 다중 파일 프로그래밍(교재 13주차)

```
합계 : 55
```

보다 빠르게! register 변수

- 지역 변수 선언 시 register 지역 변수로 선언 가능
  - 컴파일러에게 해당 변수를 RAM이 아닌 레지스터에 올려서 사용하도록 요청 → 실행 시 레지스터를 사용하지 않을 수도 있음
    - register : CPU 내에 존재하는 가장 빠른 메모리 장치

```
static int result = 0;

void Sum(int num)
{
    result += num;
}

int main(void)
{
    register int i;

    for (i = 1; i <= 10; i++)
        Sum(i);
    printf("합계 : %d \n", result);
}
```

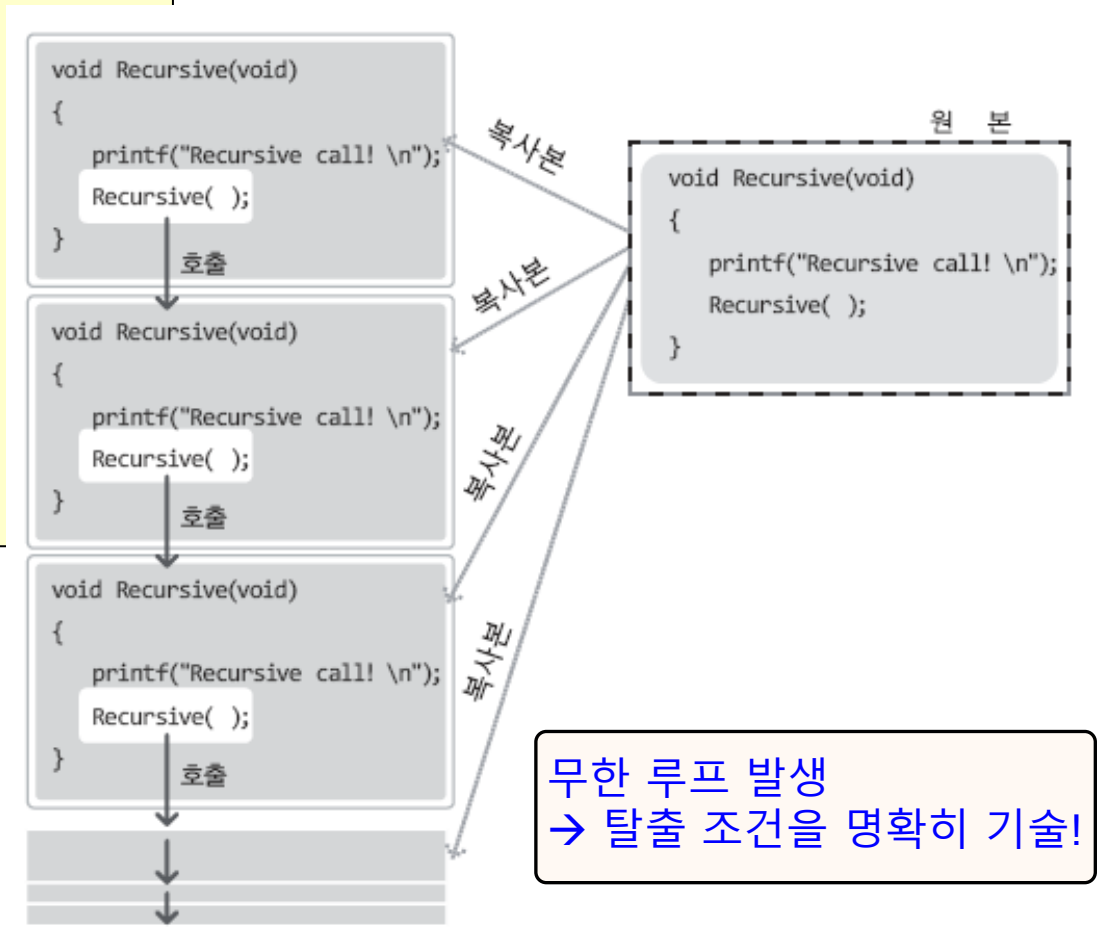
register 변수 선언의 의미
→ 이 변수는 자주 사용하는 변수니까 레지스터에 저장하는 것이 성능 향상에 도움이 될 거야!

일반적인 응용 프로그램 작성 시 자주 사용하지는 않음

재귀 함수에 대한 기본적인 이해

- 재귀 함수 : 자기 자신을 다시 호출하는 형태의 함수

```
void Recursive(void)
{
    printf("Recursive Call!
    Recursive();
}

int main(void)
{
    Recursive();
}
```



몇 번의 함수 호출 후 다시 이전의 함수가 호출되는 경우도 재귀 함수의 일종임

무한 루프 발생
→ 탈출 조건을 명확히 기술!

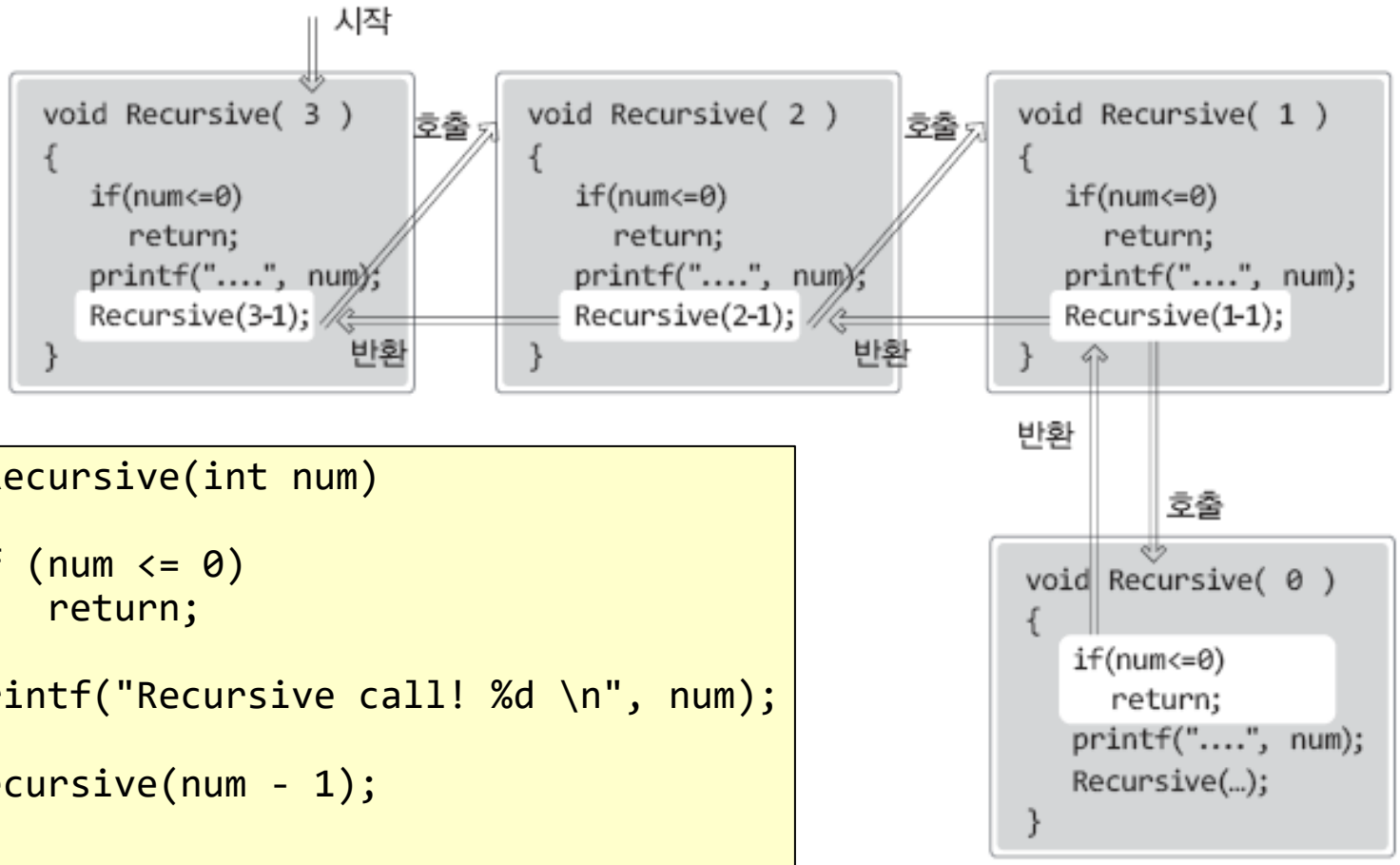
탈출 조건이 존재하는 재귀함수의 예 (1)

- Recursive(3); // 3회 호출

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

    printf("Recursive call! %d \n", num);

    Recursive(num - 1);
}

int main(void)
{
    Recursive(3);
}
```

```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```

탈출 조건이 존재하는 재귀함수의 예 (2)




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

    printf("Recursive call! %d \n", num);

    Recursive(num - 1);
}

int main(void)
{
    Recursive(3);
}
```

호출 순서의 역순으로
반환 수행

재귀 함수의 디자인 사례 (1)

- 팩토리얼(factorial) 계산을 위한 알고리즘

$$n! = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \times \cdots \times 2 \times 1}_{(n-1)!}$$

$$n! = n \times (n-1)!$$

팩토리얼에 대한 수학적 표현

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

n x f(n-1) . . . . n>=1 에 대한 코드 구현

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

f(n)=1 에 대한 코드 구현

```
if(n==0)
    return 1;
```

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

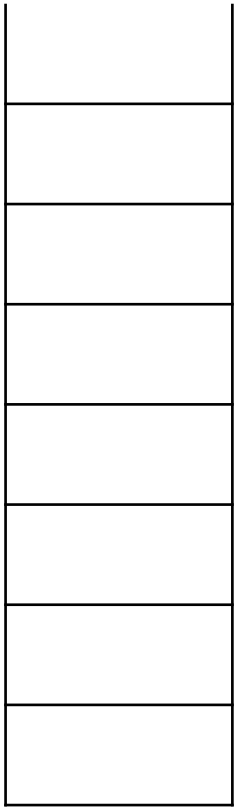
재귀 함수의 디자인 사례 (2)

```
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));
}
```

```
1! = 1
2! = 2
3! = 6
4! = 24
9! = 362880
```

```
int fact(int n)        // n!을 계산해서 반환
{
    return n == 0 ? 1 : n * fact(n - 1);
}
```

입력값 n이 0이면 1을 반환하고, 그렇지 않으면 n*fact(n-1)을 반환한다.
절차와 값을 분리해서 생각해야 함!



(지역변수)
스택

```
int main(void)
{
    fact(3);
}
```

이번 장에서 배운 것

- 함수 정의는 이름, 매개변수, 반환 타입, 함수 내용으로 이루어진다.
- 함수를 사용하기 위해서는 해당 함수를 호출해야 한다. 이때 해당 함수에 대한 선언이 앞에 나와 있어야 한다.
- 함수 정의는 그 자체로 선언이며, 이외에 함수 원형(=함수 프로토타입)을 통해 함수를 선언할 수 있다.
- 변수의 종류에는 지역 변수, 전역 변수, static 지역 변수, static 전역 변수, register 변수가 있다.
- 자기 자신을 호출하는 함수를 재귀 함수라 하고 재귀 함수를 작성할 때는 탈출 조건에 유의해야 한다.