6장. 함수

박 종 혁 교수
서울과학기술대학교 컴퓨터공학과
UCS Lab
Tel: 970-6702
Email: jhpark1@seoultech.ac.kr

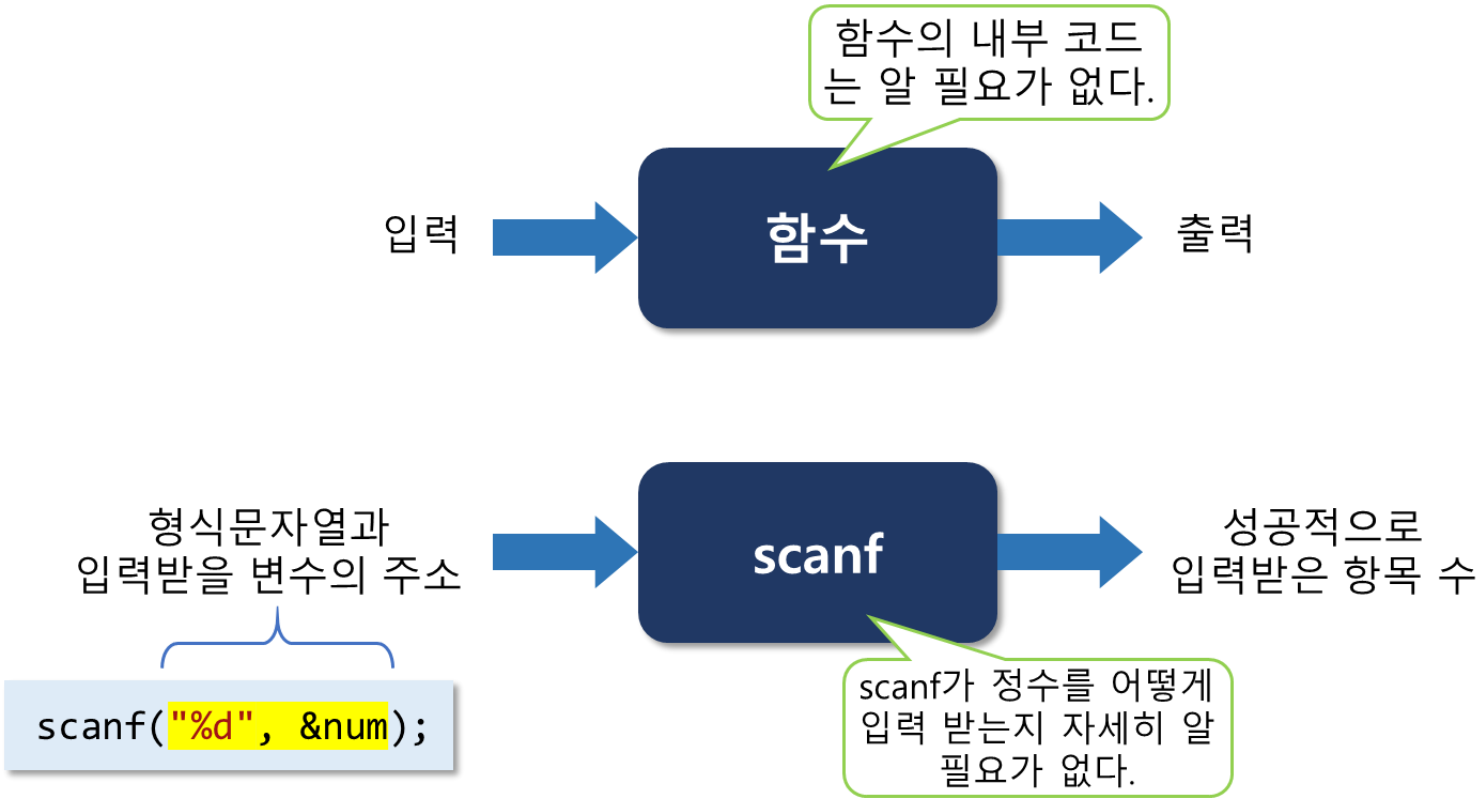
목차

함수란

❖특정 기능을 제공하는 일련의 코드를 묶어서 이름을 붙인 것

❖일종의 블랙 박스

- 전달하는 인자의 의미와 호출 결과만 알면 된다



함수란

❖하향식 프로그래밍 기법

❖프로그램은 하나 이상의 함수로 구성됨

❖함수 정의

- 함수가 수행할 일을 기술한 C 코드

❖함수 정의의 일반적인 형식

```
type function_name( parameter list )
{
 declarations
 statements
}
```

함수의 필요성

코드가 길고 복잡해서 알아보기 어렵다.

```c
#include <stdio.h>

int main(void)
{
  int amount = 10;
  int price = 1000;
  int total = amount * price;
  int i;

  for (i = 0; i < 30; i++)
    printf("-");
  printf("\n");

  printf("수량 단가 합계\n");

  for (i = 0; i < 24; i++)
    printf("*");
  printf("\n");

  printf("%d %d %d\n",
    amount, price, total);

  for (i = 0; i < 30; i++)
    printf("-");
  printf("\n");

  return 0;
}
```

'-'를 30개 출력해서 선 그리는 코드

코드를 복사해서 적당히 고쳐준다.

'*'를 24개 출력해서 선 그리는 코드

'-'를 30개 출력해서 선 그리는 코드

비슷한 일을 하는 코드가 여러 번 중복된다.

ch를 len개 출력해서 선 그리는 함수

```c
#include <stdio.h>

void draw_line(char ch, int len)
{
  int i;
  for (i = 0; i < len; i++)
    printf("%c", ch);
  printf("\n");
}

int main(void)
{
  int amount = 10;
  int price = 1000;
  int total = amount * price;

  draw_line('-', 30);

  printf("수량 단가 합계\n");

  draw_line('*', 24);

  printf("%d %d %d\n",
    amount, price, total);

  draw_line('-', 30);

  return 0;
}
```

선 그리기 코드는 한 번만 작성하면 된다.

코드가 간결하고 알아보기 쉽다.

필요할 때마다 함수를 호출한다.

함수를 사용할 때의 장점

- 코드가 중복되지 않으므로 **간결하고 알아보기 쉽다**
- 한 번 작성해둔 코드를 여러 번 사용하므로 **코드의 재사용성이 높다**
- 기능 위주로 함수를 작성해서 사용하므로 **프로그램의 모듈화**가 증대된다
- 함수 코드를 수정하더라도 함수를 호출하는 부분은 수정할 필요가 없으므로 **프로그램을 유지 보수하기 쉽다**

함수의 종류

종류	특징	예
진입점 함수	프로그램이 시작될 때 운영체제에 의해 호출된다.	`int main(void)` `{ … }`

라이브러리 함수	입출력과 같은 고유의 기능을 제공한다.
	라이브러리가 제공하므로 만들 필요는 없고, 사용만 하면 된다.
	라이브러리 함수를 호출하려면 라이브러리 헤더가 필요하다.
		`#include <stdio.h>`
		
		`char ch;`
		`scanf("%c", &ch);`
		`printf("%c", ch);`

사용자 정의 함수	프로그래머가 직접 정의하는 함수이다.
	프로그램에서 특정 기능을 제공하는 코드 부분을 묶어서 함수로 만들어 두고 사용한다.
		`void draw_line(`
		`  char ch,`
		`  int length)`
		`{ … }`

함수의 요건

- 함수의 **정의(definition)** : 함수가 수행할 내용을 기술
- 함수의 **호출(call)** : 이미 정의된 함수를 사용
- 함수의 **선언(declaration)** : 사용될 함수에 대한 정보를 미리 제공

구분	내용	예

함수의 정의	리턴형, 함수 이름, ( ) 안에 매개변수 목록을 써준 다음 { } 안에 실제로 함수가 처리할 내용을 기술한다.	`double get_area(double radius)`
		`{`
		`  ⋮`
		`}`

함수의 호출	앞에서 선언되거나 정의된 함수를 이용한다. 인자를 넘겨주고 리턴 값을 받아올 수 있다.	`printf("%f", get_area(i));`
함수의 선언	함수 호출에 필요한 리턴형, 함수 이름, 매개변수 정보를 알려준다.	`double get_area(double radius);`

함수의 정의

형식

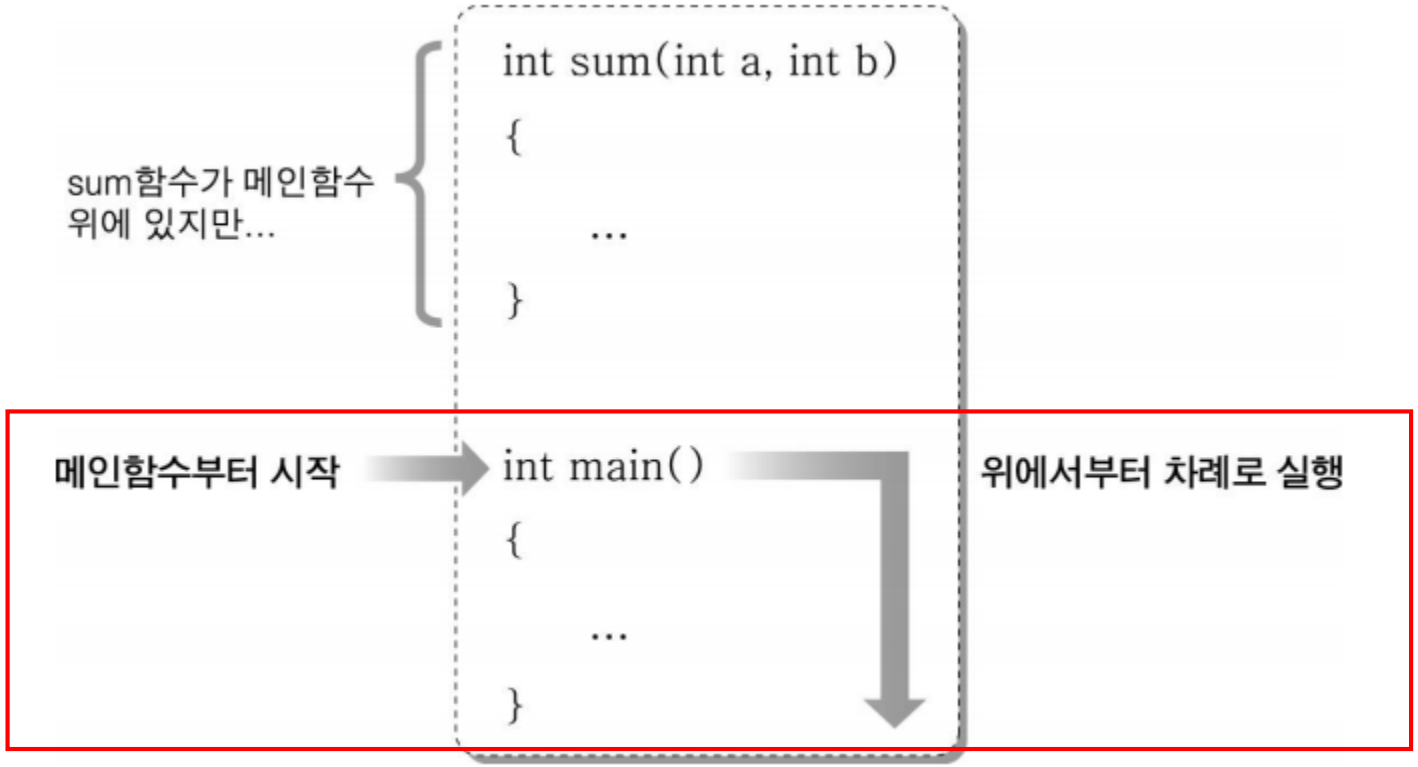
```
리턴형  함수명(매개변수목록)
{
    함수가 처리할 내용
}
```

사용예

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

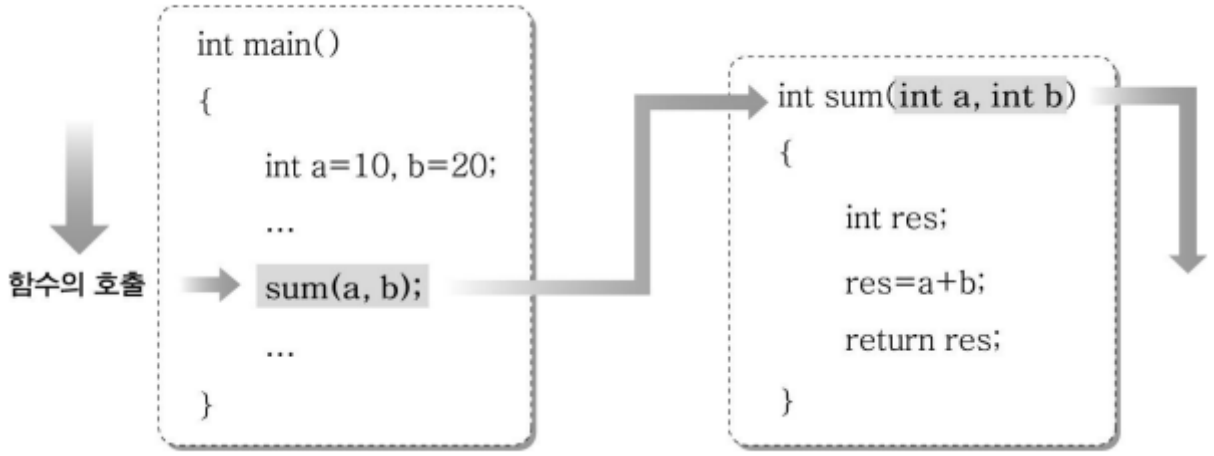
함수의 실행과정

- 함수는 호출하기 전에 정의되어 있어야 한다
- 다른 함수가 먼저 있어도 프로그램은 항상 메인함수부터 시작된다

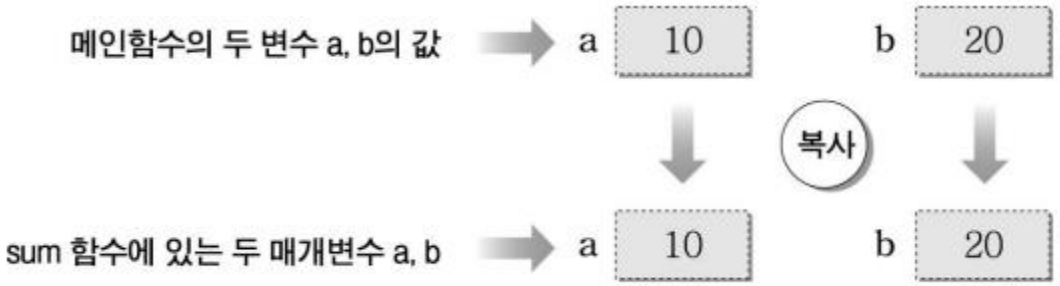


함수의 실행과정

- 메인함수의 실행 중에 다른 함수를 호출하면 그 때 함수가 실행된다

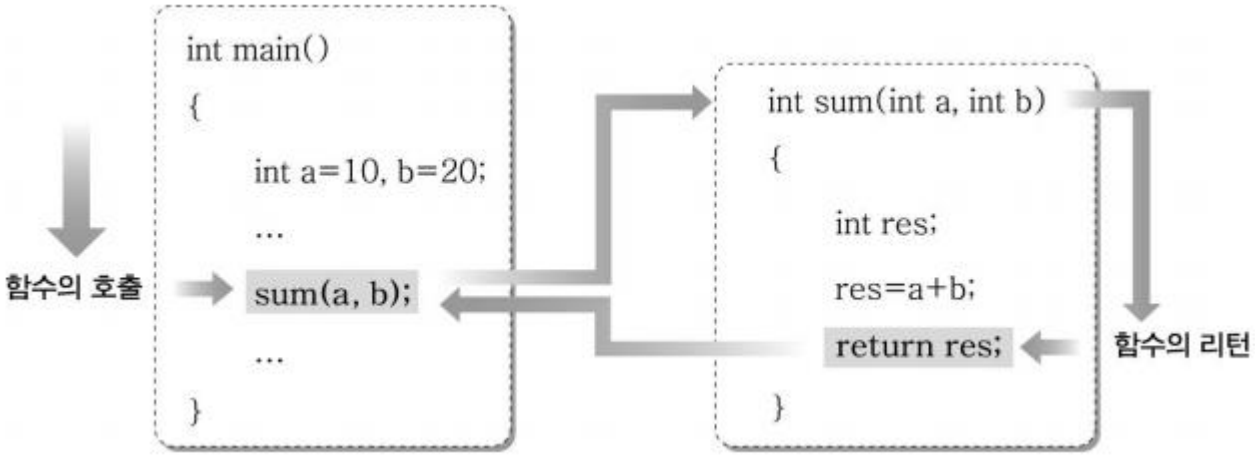


- 함수가 호출될때 전달인자는 매개변수에 복사된다



함수의 실행과정

- 함수가 실행을 마치고 리턴할 때는 제어와 함께 리턴값도 돌려준다



- 함수가 리턴하는 값은 복사되어 임시기억공간에 저장되며, 이 값은 따로 저장하지 않으면 버려지므로 다른 변수에 저장해서 사용한다

```
int res;                  //sum 함수가 리턴하는 값이 int형이므로 int형 변수를 선언한다.
res = sum(10,20);         //sum 함수가 리턴하는 값을 저장한다.
```

제어의 흐름 예제

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

제어의 흐름

```
#include <stdio.h>
int test(int);
int main(void)
{
        int b, a;

        a = 5;
        b = test(a);
        printf("test is %d", b);
        return 0;
}
```

```
int test(int c)
{
        c = c + 10;
        return 1;
}

int printf( . . . )
{
        . . . . .
}
```

함수의 리턴형

- 함수가 처리 결과로 리턴하는 값의 데이터형

```
int add(int x, int y) { … }              // 정수 값을 리턴하는 함수
double get_area(double radius) { … }     // 실수 값을 리턴하는 함수
```

- 함수가 처리 결과로 리턴하는 값이 없을 때는 void라고 적어준다

```
void draw_line(char ch, int len) { … }          // 리턴 값이 없는 함수
```

- 함수는 반드시 하나의 값만 리턴할 수 있다

함수의 이름 (1/2)

- 어떤 일을 하는 함수인지 명확하게 알 수 있는 이름을 사용한다

```
void f1(void) { … }          // 어떤 일을 하는 함수인지 알 수 없다.
void play_video(void) { … }  // 동영상을 재생하는 함수라는 것을 알 수 있다.
```

- 식별자를 만드는 규칙에 따라서 정해야 한다

```
int get_factorial(int num) { … }      // 팩토리얼을 구하는 함수
void open_file(void) { … }            // 파일 열기를 수행하는 함수
void read_data(void) { … }            // 데이터 읽기를 수행하는 함수

int GetFactorial(int num) { … }       // 팩토리얼을 구하는 함수
void OpenFile(void) { … }             // 파일 열기를 수행하는 함수
void ReadData(void) { … }             // 데이터 읽기를 수행하는 함수
```

소문자로 된 함수 이름

대소문자를 혼용하는 함수 이름

- 일관성 있는 이름을 사용하는 것이 좋다

함수의 이름 (2/2)

❖서로 다른 함수가 같은 이름을 사용할 수 없다

```
// ch를 len개 출력해서 선 그리는 함수
void line(char ch, int len) { … }

// (x1,y1)~(x2,y2)사이의 선의 길이를 구하는 함수
double line(int x1, int y1, int x2, int y2) { … }
```

같은 이름을 사용해서는 안된다.

❖명명 규칙(Naming Convention)

- 식별자로 사용되는 이름을 정할 때 적용되는 규칙
- 프로그래머가 직접 자신만의 명명 규칙을 정하고 따르면 코드의 가독성을 높일 수 있다

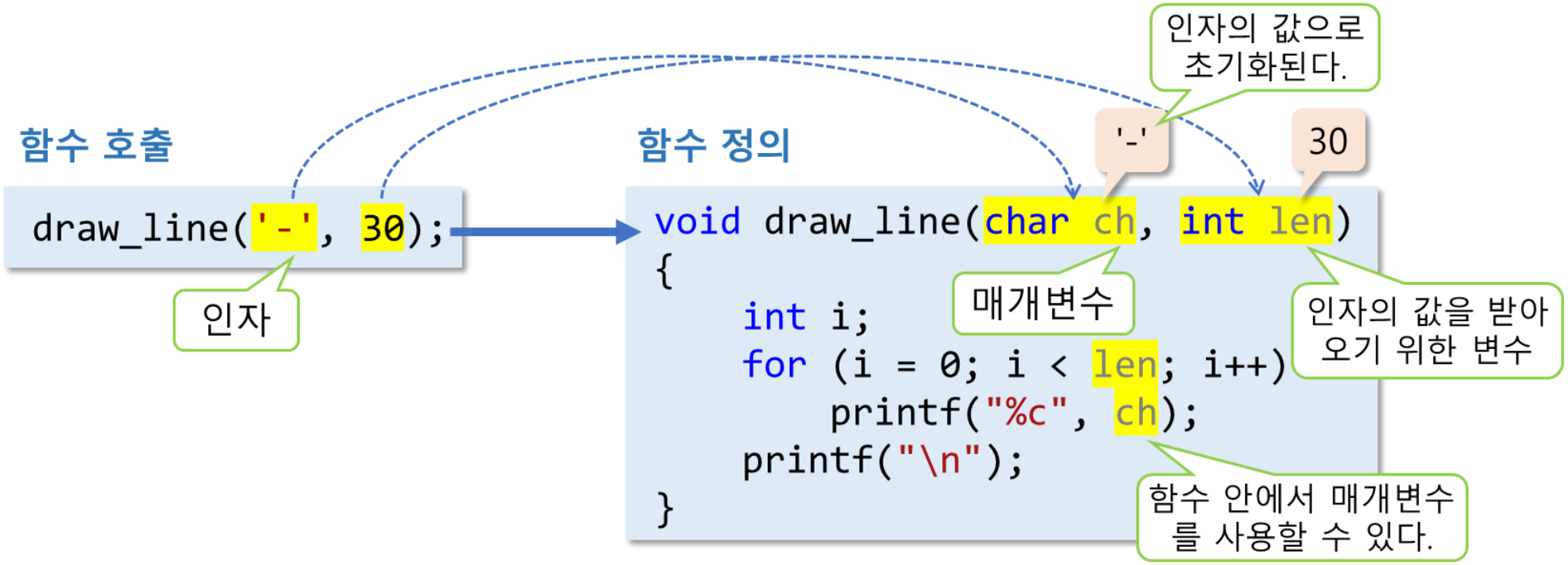
매개변수 목록 (1/2)

❖매개변수(parameter)

- 함수를 호출한 곳에서 함수 안으로 전달되는 값을 보관하기 위한 변수

❖인자, 인수(argument)

- 함수를 호출할 때, 실제로 전달되는 값



매개변수 목록 (2/2)

❖매개변수의 개수에는 제한이 없다

- 매개변수가 없을 때는 void를 써주는데, 이때의 void는 생략할 수 있다

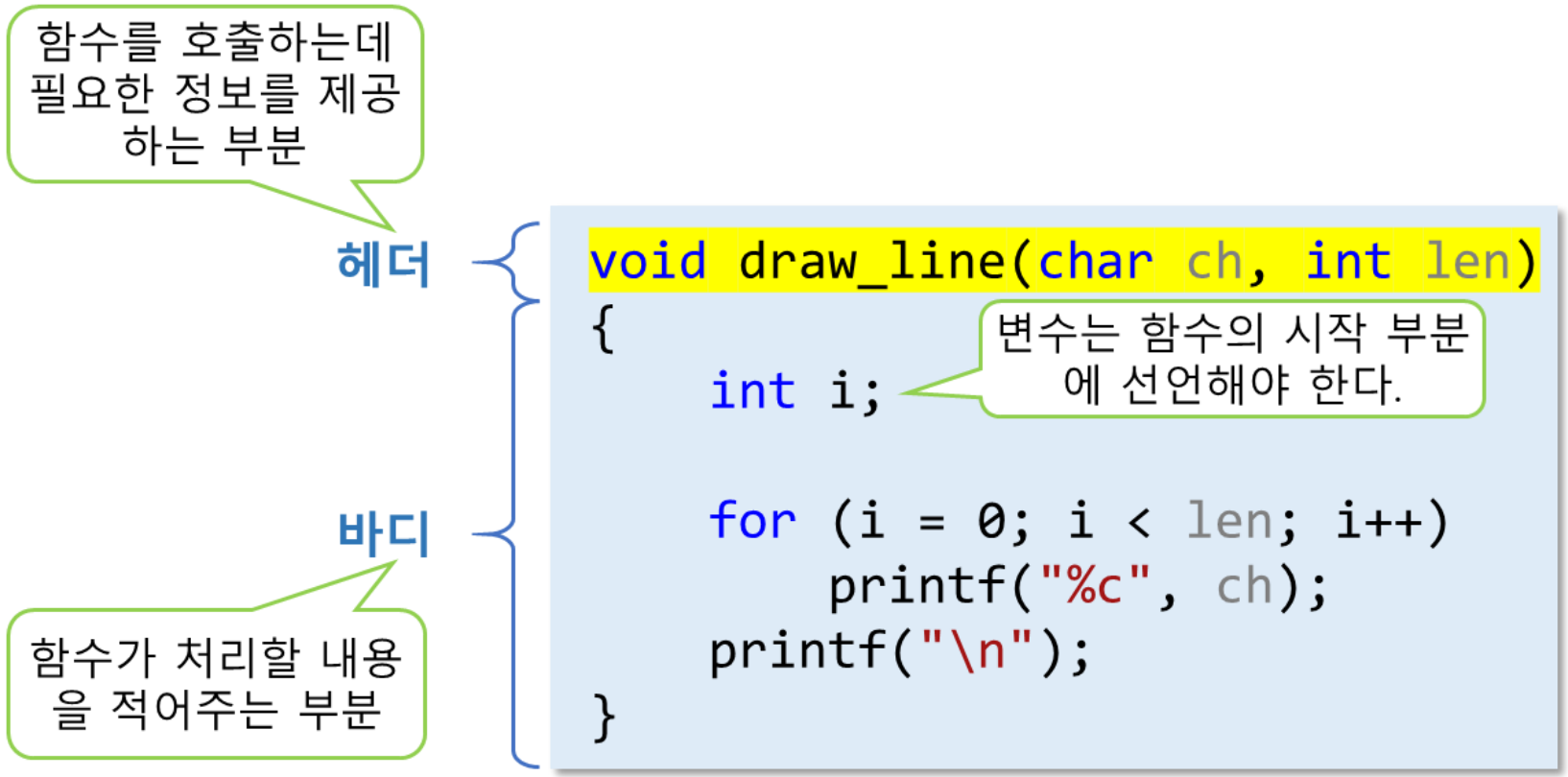
```
void play_video(void) { … }            // 매개변수가 없는 함수
int get_factorial(int num) { … }       // 매개변수가 1개인 함수
int add(int x, int y) { … }            // 매개변수가 2개인 함수

void open_file() {}                    // 매개변수가 없을 때 void를 생략할 수 있다.
```

❖함수를 정의할 때, 매개변수의 데이터형을 생략하거나 매개변수의 이름을 생략하면 컴파일 에러가 발생한다

```
void print_sum(count) { … }            // 매개변수의 데이터형을 생략하면 컴파일 에러
void draw_line(char ch, int) { … }     // 매개변수의 이름을 생략하면 컴파일 에러
```

함수의 내용

- 함수의 정의는 헤더(header)와 바디(body)로 구성된다



함수 헤더

❖헤더
- 함수 정의에서 첫 번째 여는 중괄호의 앞 부분 type

❖**function_name(parameter list)**
- **type**
  - 함수가 리턴하는 값의 형
  - 컴파일러는 필요하다면, 함수의 리턴 값을 이 type으로 변환함
  - 이것이 void이면 리턴하는 값이 없다는 것을 나타냄
- **parameter list**
  - 이 함수가 가지는 인자의 목록
  - 이 함수를 호출할 때에는 이 list에 맞게 호출해야 함
  - 이것이 void이면 인자를 갖지 않음을 나타냄

함수 몸체

❖몸체

- 함수 정의에서 중괄호 사이에 있는 문장들

❖예제

```
int factorial(int n)                    /* header */
{                                       /* body starts here */
        int i, product = 1;
        for (i = 2; i <= n; ++i)
                product *= i;
        return product;
}
```

리턴 값과 매개변수가 없는 함수

- 리턴형과 매개변수 목록을 void로 지정한다
- void형의 함수를 호출하면 정해진 코드를 수행하고, 함수의 끝을 만나면 리턴한다

```
void hi(void)          // 리턴형과 매개변수가 없는 함수
{
    printf("Hi! Let's enjoy C programming.\n");
}                      // 함수의 끝을 만나면 리턴한다.

void bye(void) { printf("Bye!\n"); }    // 간단한 함수는 한 줄로 작성할 수 있다.
```

리턴 값은 없고 매개변수만 있는 함수 (1/2)

```
void draw_line(char ch, int len)        // ch는 출력에 사용할 문자, len은 문자의 개수
{
    int i;
    for (i = 0; i < len; i++) // ch를 len개 출력한다.
        printf("%c", ch);
    printf("\n");
}
```

선 그리는 기능

리턴 값은 없고 매개변수만 있는 함수 (2/2)

```c
void print_sum(int count)     // count는 입력받을 정수의 개수
{
    int i;
    int num;                  // 입력받을 정수를 저장할 변수
    int sum = 0;              // 합계를 저장할 변수

    printf("%d개의 정수? ", count);
    for (i = 0; i < count; i++)        // 정수를 count개 입력받아 합계를 구한다.
    {
        scanf("%d", &num);
        sum += num;
    }
    printf("합계 : %d\n", sum);        // sum을 출력할 뿐 그 값을 리턴하지는 않는다.
}
```

count 개의 정수를 입력받아 합계를 구해서 출력하는 함수

리턴 값과 매개변수가 있는 함수

- 리턴형이 있는 함수에서 return문을 생략하면 안된다

num 팩토리얼을 구하는 함수

```
int get_factorial(int num)    // int형의 값을 리턴한다.
{
    int i;
    int result = 1;

    for (i = 1; i <= num; i++)          // num!을 구한다.
        result *= i;
    return result;  // 구한 num! 값을 리턴한다.
}
```

원의 면적을 구하는 함수

```
double get_area(double radius)
{
    const double PI = 3.14159265359;
    return PI * radius * radius;
}
```

기존의 코드로부터 함수를 정의하는 과정

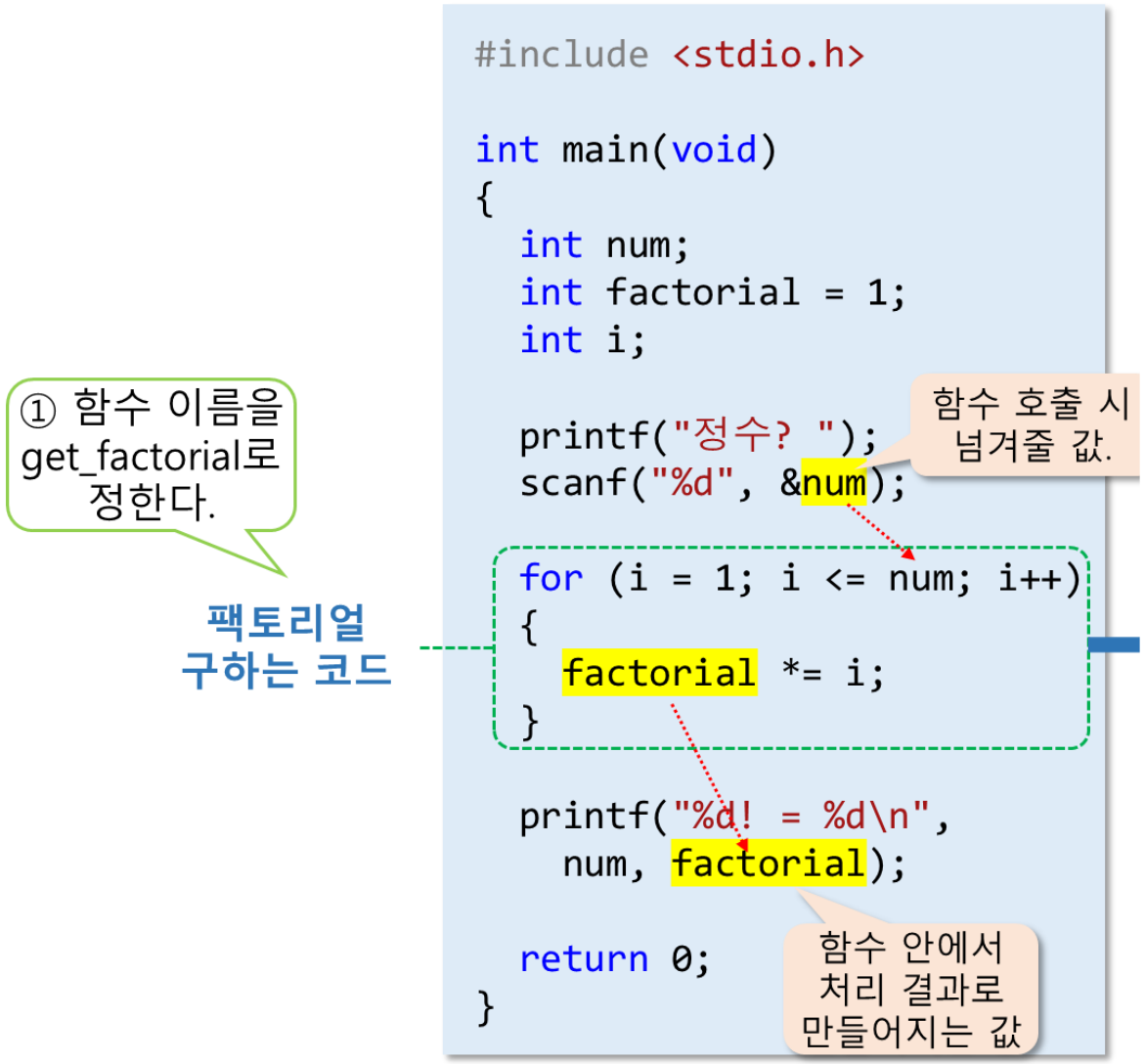
```c
#include <stdio.h>

int main(void)
{
  int num;
  int factorial = 1;
  int i;

  printf("정수? ");
  scanf("%d", &num);

  for (i = 1; i <= num; i++)
  {
    factorial *= i;
  }

  printf("%d! = %d\n",
    num, factorial);

  return 0;
}
```

함수 호출 시 넘겨줄 값.

함수 안에서 처리 결과로 만들어지는 값

어떤 내용을 어떻게 만들것인가 ?

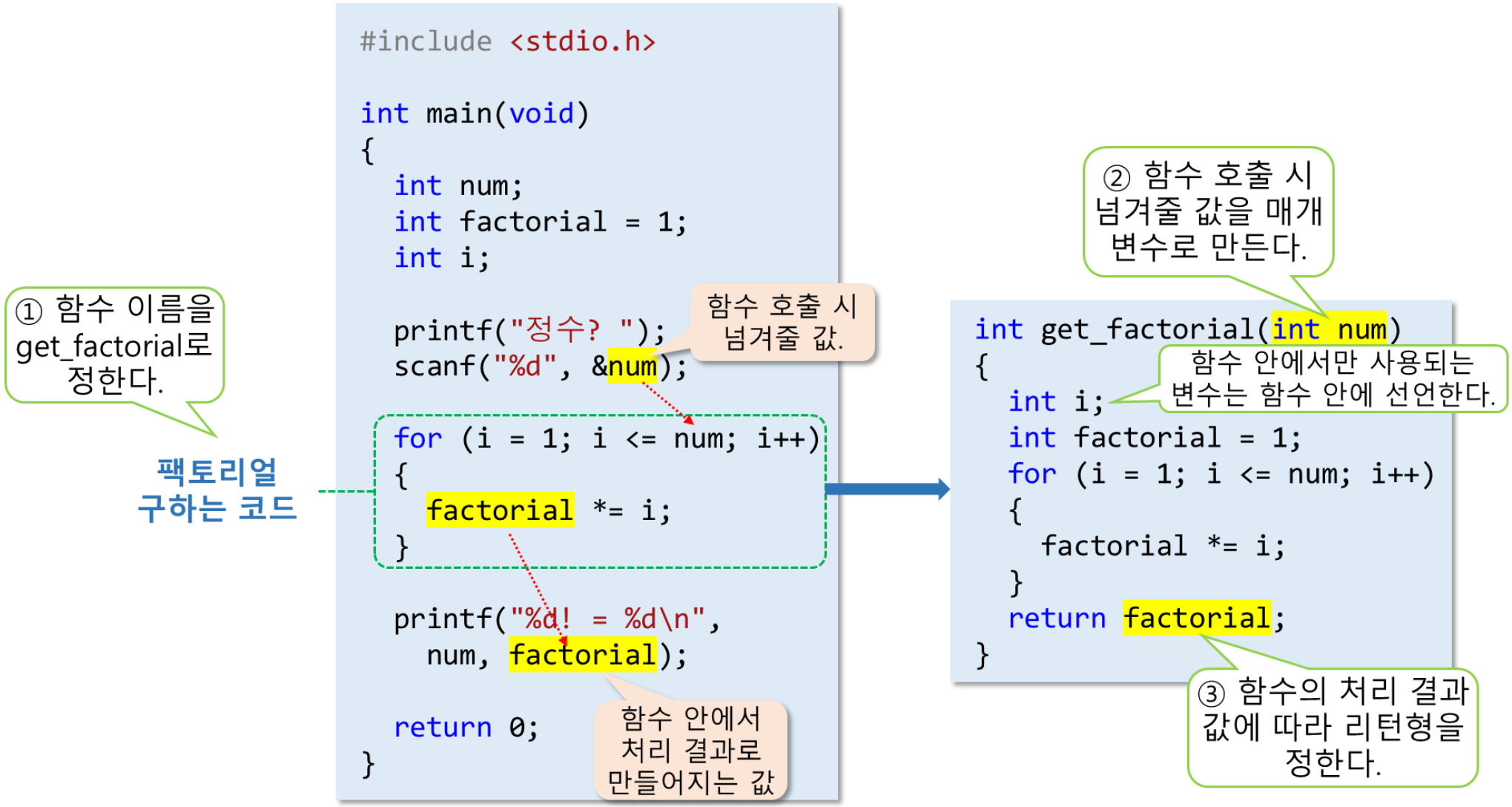
기존의 코드로부터 함수를 정의하는 과정



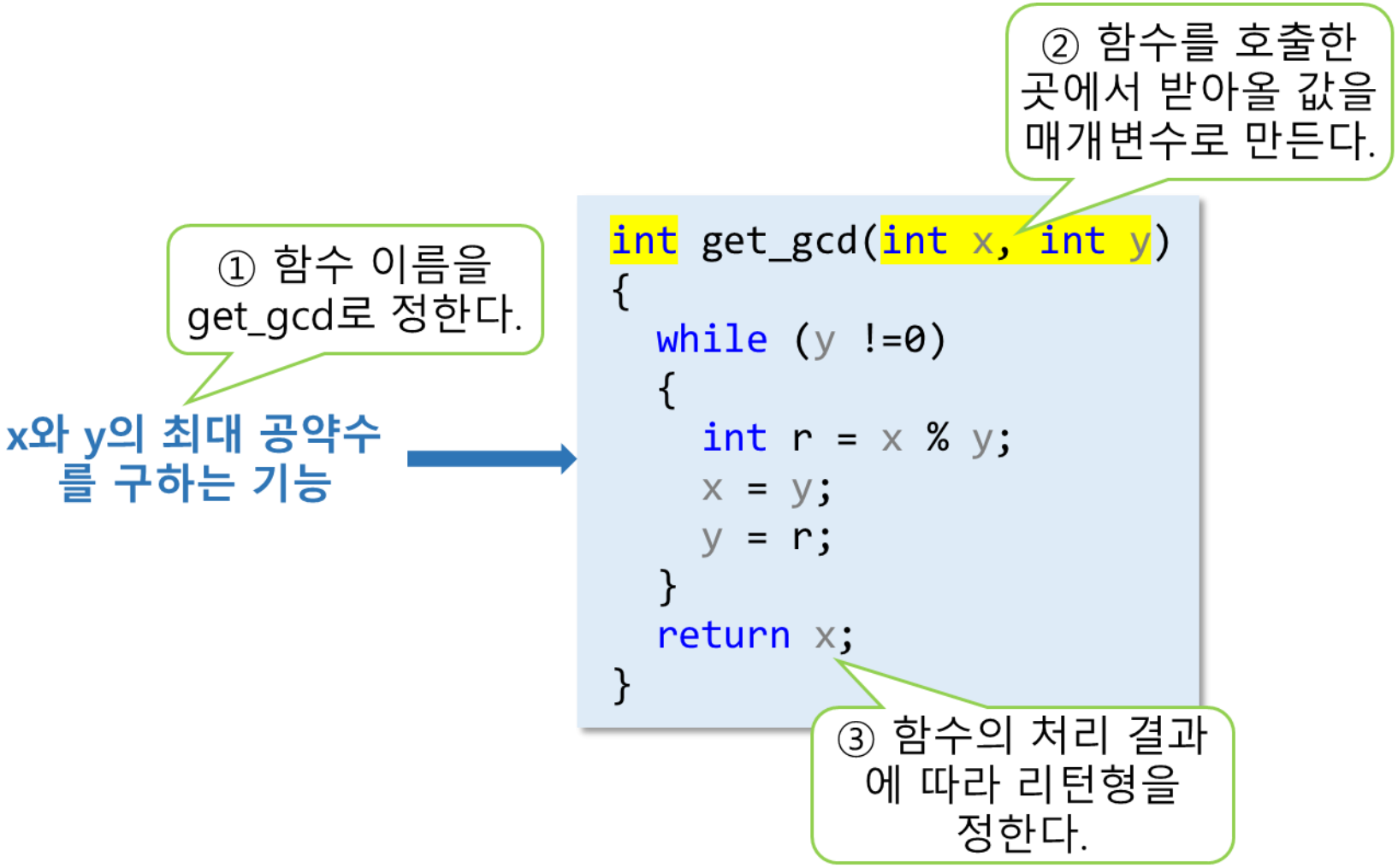

```c
#include <stdio.h>

int main(void)
{
  int num;
  int factorial = 1;
  int i;

  printf("정수? ");
  scanf("%d", &num);

  for (i = 1; i <= num; i++)
  {
    factorial *= i;
  }

  printf("%d! = %d\n",
    num, factorial);

  return 0;
}
```

어떤 내용을 어떻게 만들것인가 ?

기존의 코드로부터 함수를 정의하는 과정



처음부터 함수를 정의하는 과정



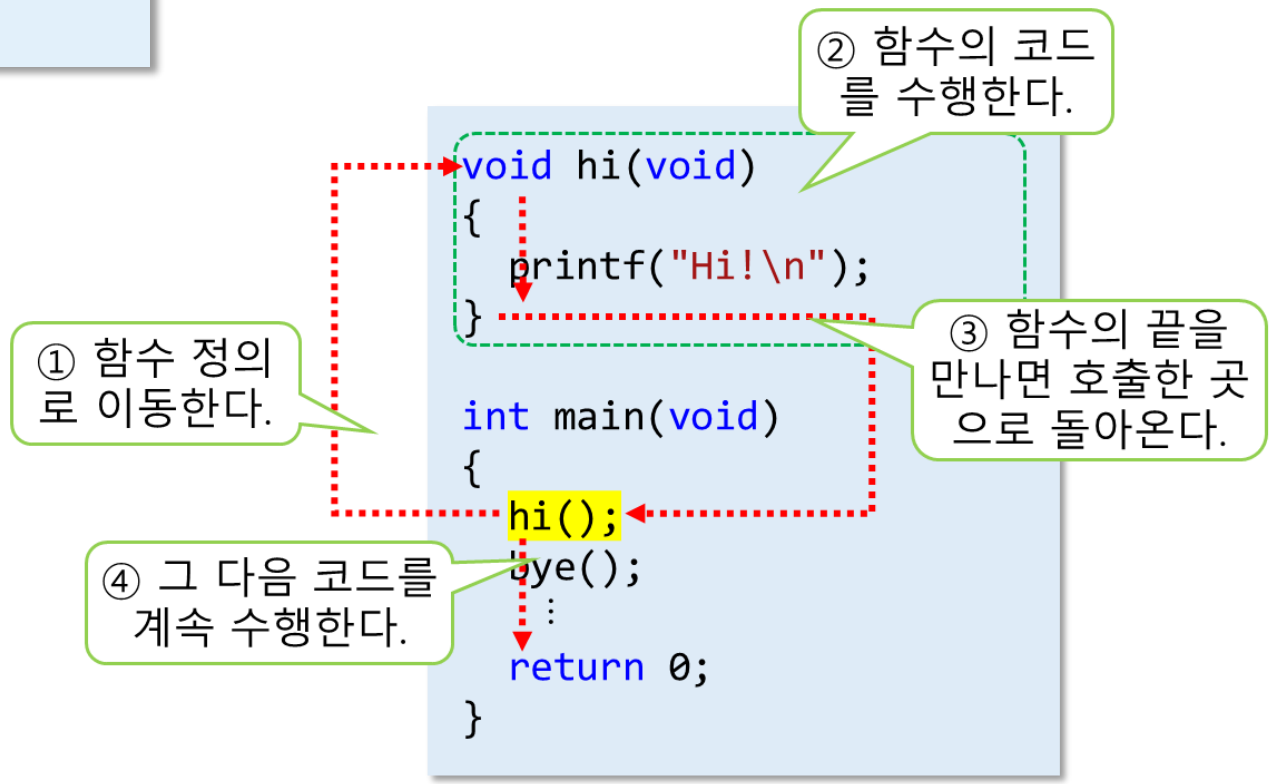
함수의 호출

- 이미 만들어진 함수를 불러 쓰는 것
- 함수 이름 다음에 ( )를 쓰고, ( ) 안에 함수의 인자를 써준다
- 함수를 호출할 때 넘겨준 인자가 매개변수로 전달된다

리턴 값과 매개변수가 없는 함수의 호출

❖hi();처럼 함수 이름 다음에 빈 괄호를 적어준다
- ( )가 없으면 함수 호출이 아니다

```
hi();  // 매개변수가 없는 함수의 호출
```



예제 : 리턴 값과 매개변수가 없는 함수의 사용 예

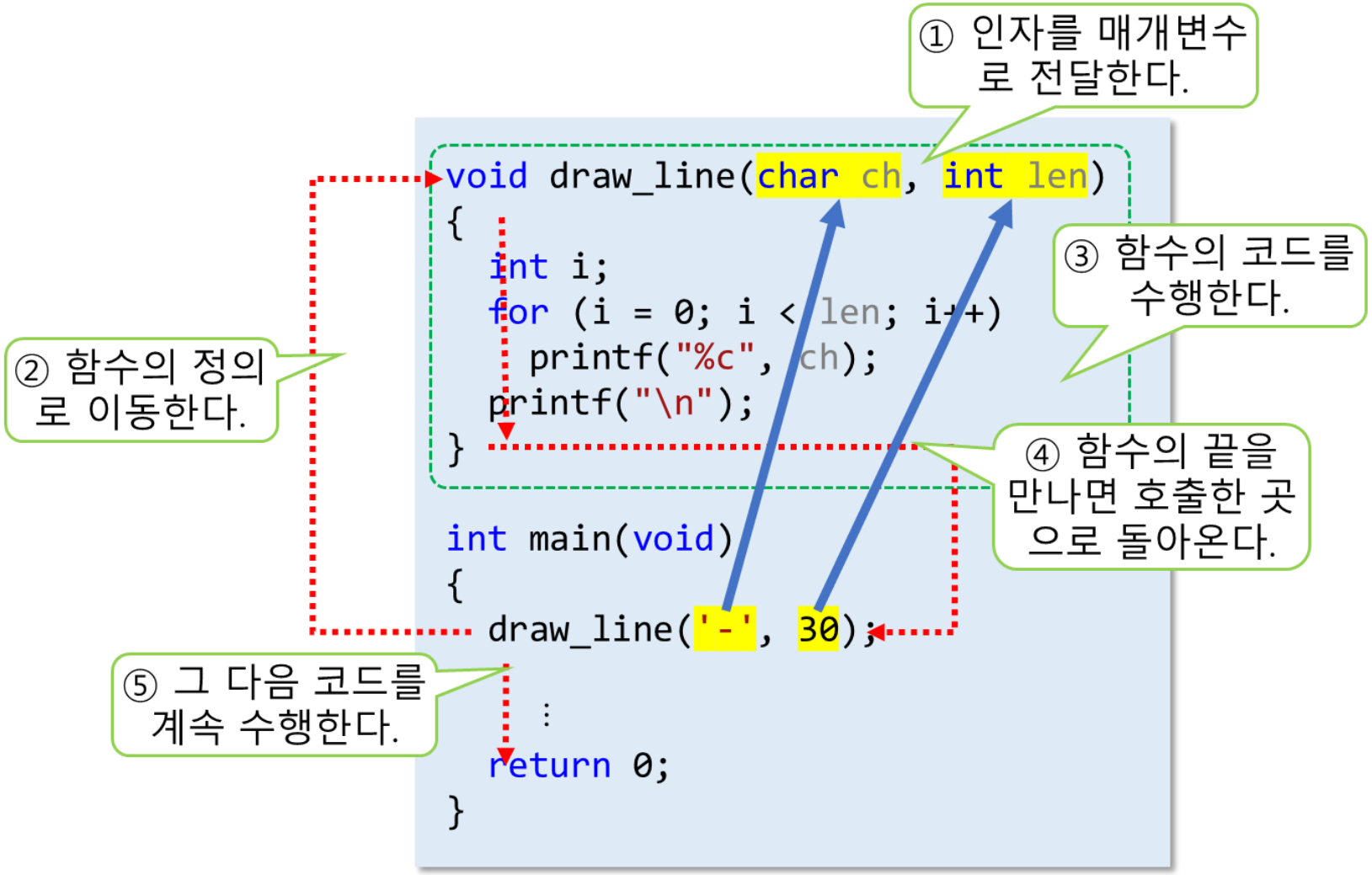
```
03  void hi(void)       // 리턴형과 매개변수가 없는 함수
04  {
05      printf("Hi! Let's enjoy C programming.\n");
06  }                   // 함수의 끝을 만나면 리턴한다.
07
08  void bye() { printf("Bye!\n"); }       // 간단한 함수는 한 줄로 작성할 수 있다.
09
10  int main(void)
11  {
12      hi();
13      bye();
14
15      hi();
16      bye();
17
18      return 0;
19  }
```

실행결과

```
Hi! Let's enjoy C programming.
Bye!
Hi! Let's enjoy C programming.
Bye!
```

같은 함수를 여러 번 호출할 수 있다.

리턴 값은 없고 매개변수만 있는 함수의 호출



예제 : 매개변수가 있는 draw_line 함수의 사용 예

```
05  void draw_line(char ch, int len)
06  {
07      int i;
08      for (i = 0; i < len; i++)
09          printf("%c", ch);
10      printf("\n");
11  }
12
13  int main(void)
14  {
15      int amount = 10;
16      int price = 1000;
17      int total = amount * price;
18      int width;      // 계산서 헤더 폭
20      draw_line('-', 30);         // '-'로 30개 길이만큼 선을 그린다.
21
22      printf("수량     단가      합계\n");
23      width = 3 + 8 + 8 + 2;    // 계산서 헤더 폭
24      draw_line('*', width);     // '*'로 width개 길이만큼 선을 그린다.
25      printf("%3d %8d %8d\n", amount, price, total);
26
27      draw_line('-', 30);         // '-'로 30개 길이만큼 선을 그린다.
28
29      return 0;
30  }
```

실행결과

```
------------------------------
수량     단가      합계
**********************
 10     1000    10000
------------------------------
```

draw_line('-',30);

draw_line('*',width);

draw_line('-',30);

인자 전달 시 주의 사항

- ( ) 안에 함수의 인자를 콤마(,)로 나열한다
  - 함수 호출 시 넘겨준 인자는 함수의 매개변수로 순서대로 전달된다
- 매개변수의 데이터형과 같은 형의 값을 인자로 전달해야 한다
- 인자와 매개변수의 데이터형과 일치하지 않으면 형 변환해서 전달한다
- 인자와 매개변수의 순서와 개수가 일치해야 한다
- 매개변수의 의미에 맞게 순서대로 인자를 전달해야 한다

```
draw_line('-', 30);
```

char형으로 형 변환

```
draw_line(101, 50);
```

인자와 매개변수의 개수가 다르므로 컴파일 에러

```
draw_line('-');     //
```

인자의 순서가 잘못된 경우

```
draw_line(30, '-');
```

예제 : 매개변수가 있는 print_sum 함수의 사용 예

```
03  void print_sum(int count)
04  {
05      int i;
06      int num;        // 입력받을 정수를 저장할 변수
07      int sum = 0;    // 합계를 저장할 변수
08
09      printf("%d개의 정수? ", count);
10      for (i = 0; i < count; i++)         // 정수를 count개 입력받아 합계를 구한다.
11      {
12          scanf("%d", &num);
13          sum += num;
14      }
15      printf("합계 : %d\n", sum);
16  }
18  int main(void)
19  {
20      int i;
21
22      for (i = 3; i < 10; i += 2)
23      {
24          print_sum(i);         // 반복문 안에서 함수를 호출할 수 있다.
25      }
26
27      return 0;
28  }
```

실행결과

```
3개의 정수? 3 5 12
합계 : 20
5개의 정수? 54 66 34 1 45
합계 : 200
7개의 정수? 8 3 42 5 6 22 45
합계 : 131
9개의 정수? 11 23 15 43 23 99 23 8 33
합계 : 278
```

print_sum(3);

print_sum(5);

print_sum(7);

print_sum(9);

리턴 값이 없는 함수 호출 시 주의 사항

❖리턴형이 void형인 함수는 리턴 값이 없으므로 수식이 아니다

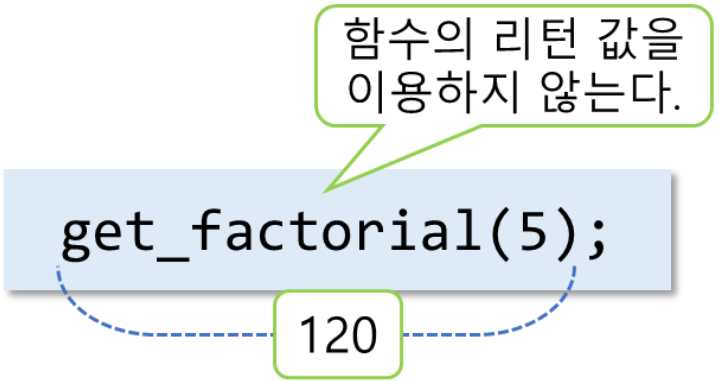
- 수식이 사용되어야 하는 곳에 리턴 값이 없는 함수 호출은 사용할 수 없다

```
printf("%d", print_sum(3));   // void형의 함수 호출은 리턴 값이 없으므로
                              // 수식으로 사용할 수 없다.
```

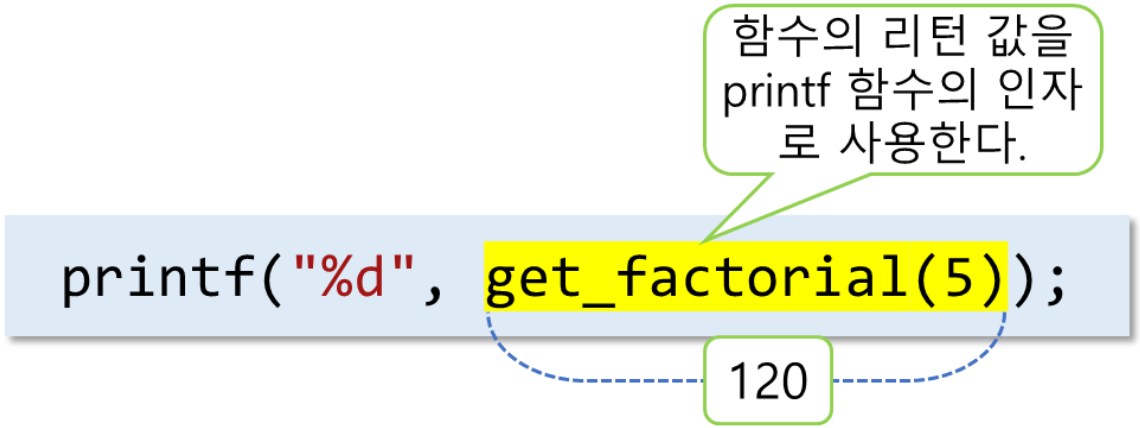
컴파일 에러

리턴 값과 매개변수가 있는 함수의 호출 (1/2)

- 함수의 리턴 값은 임시 값이므로 사용하지 않으면 사라진다
- 함수의 리턴 값은 변수에 저장할 수도 있고, 다른 수식의 일부분이나 다른 함수 호출의 인자로 사용할 수 있다

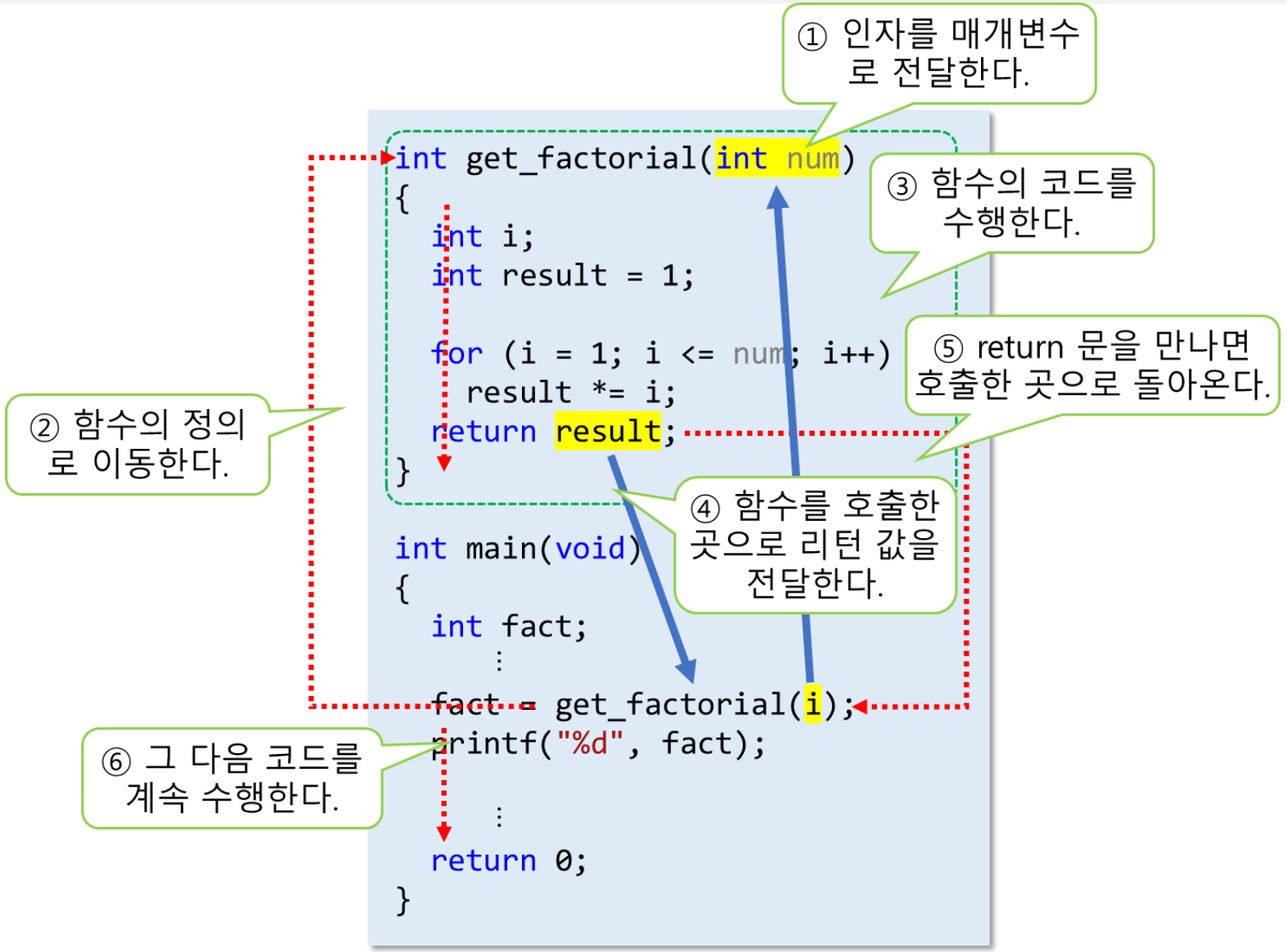


함수의 리턴 값도 임시 값이므로 변수에 저장하지 않으면 사라진다.



함수의 리턴 값을 다른 수식에 이용할 수 있다.

리턴 값과 매개변수가 있는 함수의 호출 (2/2)



예제 : get_factorial 함수의 사용 예

```
03  int get_factorial(int num)
04  {
05      int i;
06      int result = 1;
07
08      for (i = 1; i <= num; i++)  // num!을 구한다.
09          result *= i;
10      return result;  // num! 값을 리턴한다.
11  }
13  int main(void)
14  {
15      int i;
16      int fact;
17
18      for (i = 1; i <= 5; i++)
19      {
20          fact = get_factorial(i); // get_factorial(i)의 리턴 값을 저장한다.
21          printf("%2d! = %3d\n", i, fact);
22      }
23      get_factorial(5);   // 리턴 값을 어디에도 이용하지 않는다.
24
25      return 0;
26  }
```

실행결과

```
1! =   1
2! =   2
3! =   6
4! =  24
5! = 120
```

예제 : 원의 면적을 구하는 get_area 함수 사용 예

```
03  double get_area(double radius)          // 원의 면적을 구하는 함수
04  {
05      const double PI = 3.14159265359;
06      return PI * radius * radius;
07  }
08
09  int main(void)
10  {
11      int i;
12      for (i = 1; i <= 5; i++)
13      {
14          printf("반지름이 %d일 때 원의 면적: %.2f\n", i, get_area(i));
15      }
16
17      return 0;
18  }
```

실행결과

```
반지름이 1일 때 원의 면적: 3.14
반지름이 2일 때 원의 면적: 12.57
반지름이 3일 때 원의 면적: 28.27
반지름이 4일 때 원의 면적: 50.27
반지름이 5일 때 원의 면적: 78.54
```

인자 전달 과정의 의미

```
double get_area(double radius)
{
    const double PI = 3.14159265359;
    return PI * radius * radius;
}

int main(void)
{
    int i;
    for (i = 1; i <= 10; i++)
    {
        printf("%.2f", get_area(i));
    }

    return 0;
}
```

double radius = i;

인자를 매개변수의 데이터형에 맞춰 형 변환한다.

매개변수는 인자로 초기화된다.

```
double radius = i;
```

int → double

인자가 매개변수로 전달될 때 실제로 수행되는 문장

예제 : 두 정수의 최대공약수를 구하는 get_gcd 함수의 사용 예

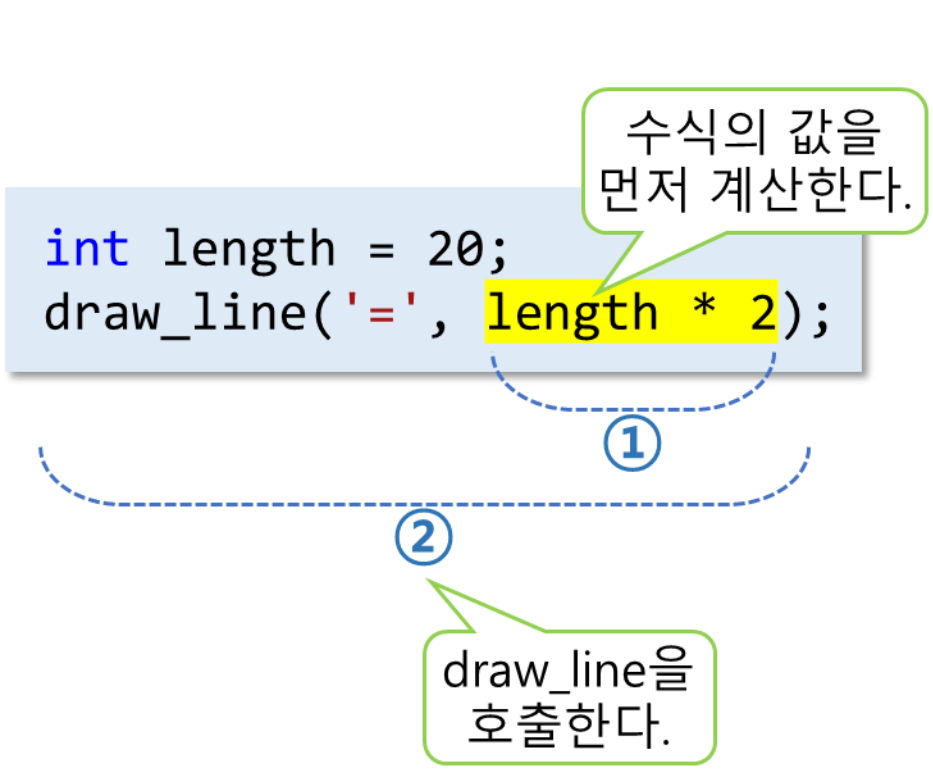
```
03  int get_gcd(int x, int y)
04  {
05      int r;
06      while (y != 0) {    // 유클리드 호제법으로 최대공약수를 구한다.
07          r = x % y;
08          x = y;
09          y = r;
10      }
11      return x;
12  }
14  int main(void)
15  {
16      int x, y;
17      int gcd;
18
19      while (1) {
20          printf("정수 2개를 입력하세요. (0 0이면 종료): ");
21          scanf("%d %d", &x, &y);
22          if (x == 0 && y == 0)
23              break;
24          gcd = get_gcd(x, y);
25          printf("%d와 %d의 GCD: %d\n", x, y, gcd);
26      }
27      return 0;
28  }
```

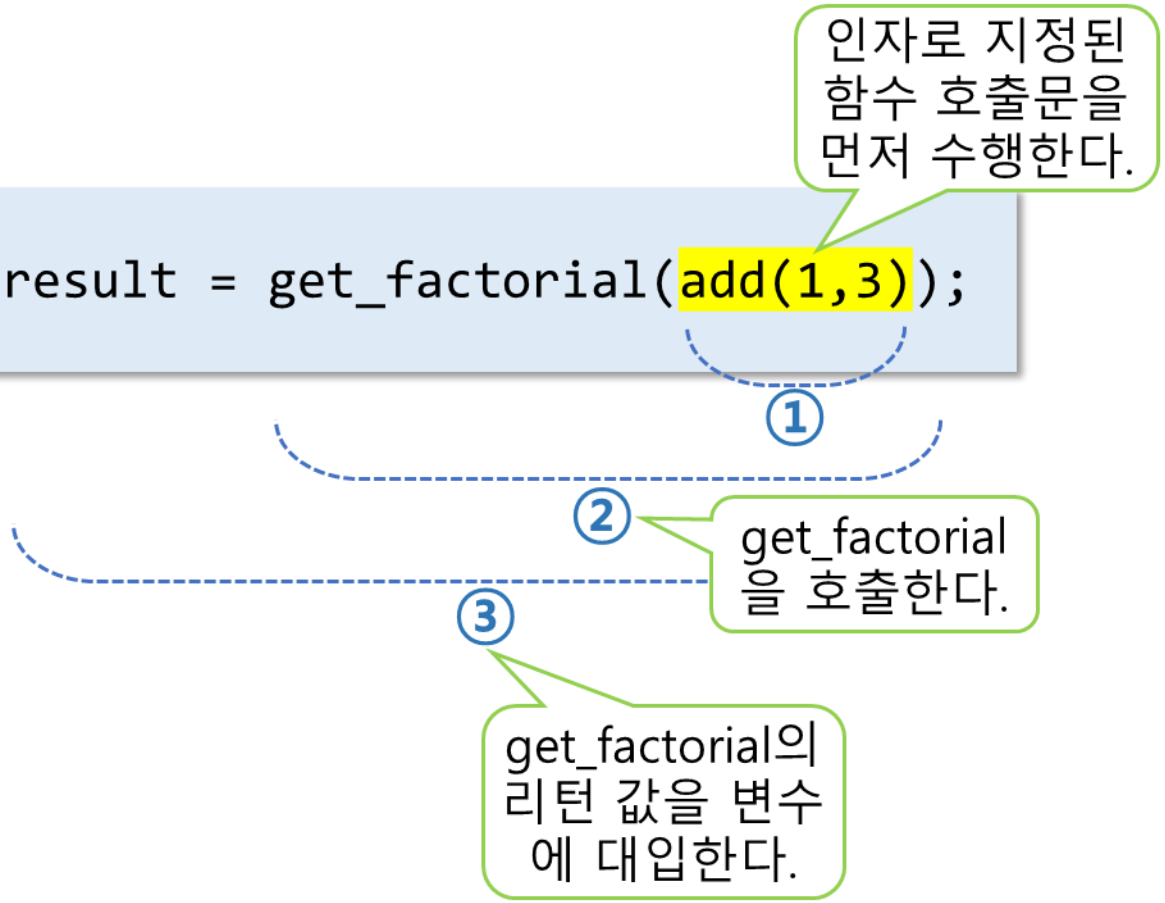
실행결과

```
정수 2개를 입력하세요. (0 0이면 종료): 10 15
10와 15의 GCD: 5
정수 2개를 입력하세요. (0 0이면 종료): 42 36
42와 36의 GCD: 6
정수 2개를 입력하세요. (0 0이면 종료): 0 0
```

함수 호출 시 주의 사항 (1/3)

- 함수의 인자도 수식이며, 항상 인자의 값을 먼저 평가한다





예제 : get_max 함수의 사용 예

```
03  int get_max(int a, int b, int c)
04  {
05      int max = a > b ? a : b;
06      max = c > max ? c : max;
07      return max;
08  }
09
10  int main(void)
11  {
12      int x, y, z;
13
14      while (1)
15      {
16          printf("정수 3개를 입력하세요 (0 0 0 입력 시 종료): ");
17          scanf("%d %d %d", &x, &y, &z);
19          if (x == 0 && y == 0 && z == 0)
20              break;
21          printf("최대값: %d\n", get_max(x, y, z));
22      }
23      return 0;
24  }
```

실행결과

```
정수 3개를 입력하세요 (0 0 0 입력 시 종료): 123 -234 5
최대값: 123
정수 3개를 입력하세요 (0 0 0 입력 시 종료): 87 333 77
최대값: 333
정수 3개를 입력하세요 (0 0 0 입력 시 종료): 0 0 0
```

함수 호출 시 주의 사항 (2/3)

- 인자의 개수와 데이터형은 매개변수와 일치해야 한다

```
int get_max(int a, int b, int c)
{
    int max = a > b ? a : b;
    max = c > max ? c : max;
    return max;
}
```

매개변수가 3개인 함수

```
get_max(10, 20);
```

인자가 매개변수보다 부족하므로 컴파일 에러

```
get_max(10, 20, 30, 40);
```

인자가 매개변수보다 많으므로 컴파일 경고

```
double max;

max = get_max(12.34, 0.5, 7.9);
```

인자를 매개변수의 데이터형에 맞춰 형 변환한다.

```
max = get_max( 12,    0,   7 );
```

12가 리턴된다.

데이터의 손실 발생

컴파일 경고

함수 호출 시 주의 사항 (3/3)

❖함수는 이름으로 구분한다
- 같은 이름의 함수를 여러 번 정의할 수 없다

같은 이름을 사용해서는 안된다.

```
int get_max(int a, int b, int c)
{
  int max = a > b ? a : b;
  max = c > max ? c : max;
  return max;
}

double get_max(double a, double b)
{
  return a > b ? a : b;
}
```

같은 이름의 함수를 재정의하면 안된다.

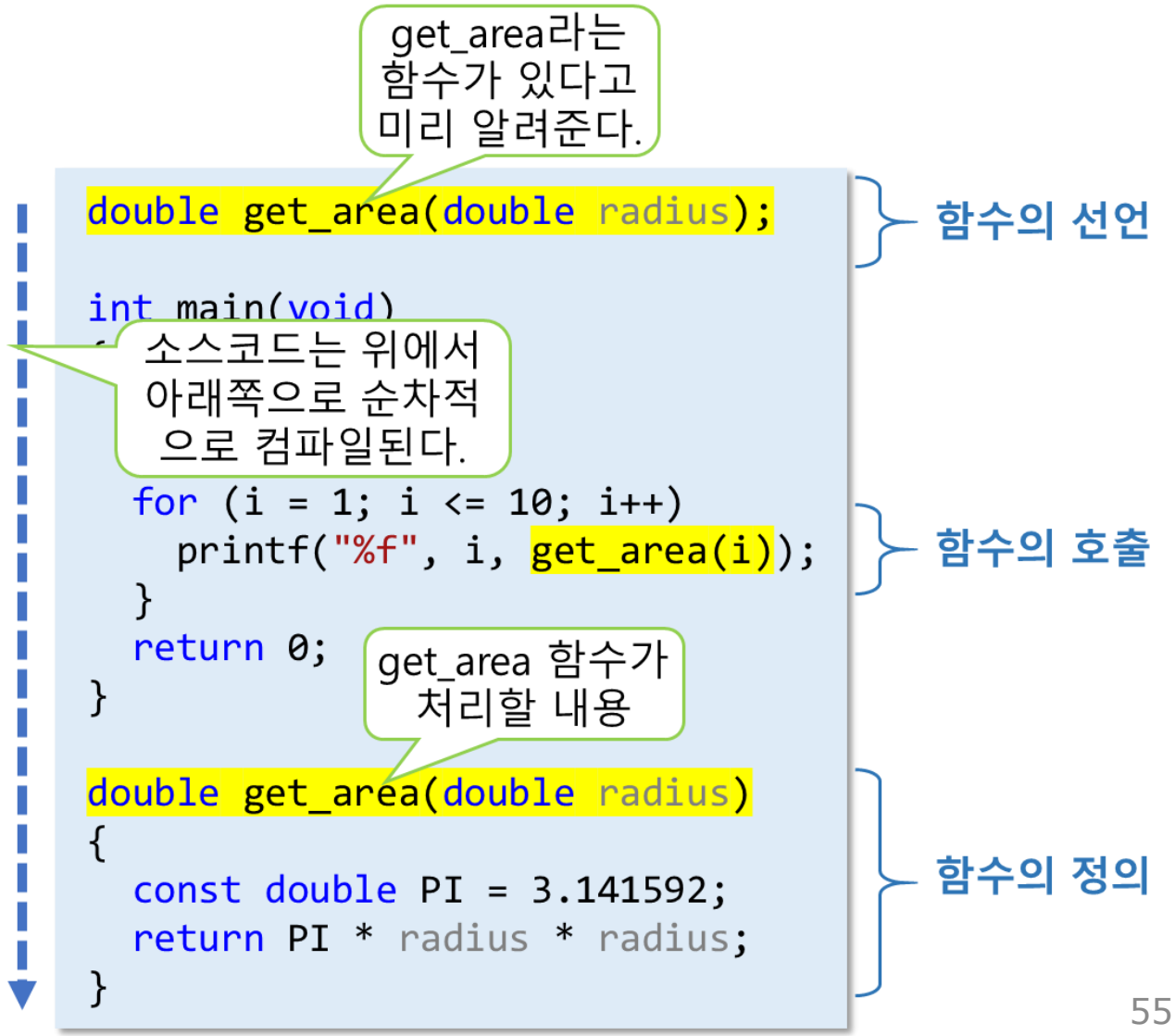
함수마다 다른 이름을 사용해야 한다.

```
int get_imax(int a, int b, int c)
{
  int max = a > b ? a : b;
  max = c > max ? c : max;
  return max;
}

double get_fmax(double a, double b)
{
  return a > b ? a : b;
}
```

반드시 다른 이름을 사용해야 한다.

함수의 선언

- 함수가 정의된 위치에 상관 없이 호출하려면 함수의 선언이 필요하다
- 함수의 리턴형, 이름, 매개변수에 대한 정보를 미리 알려준다
- 함수의 **원형(prototype)**
- 함수의 **시그니쳐(signature)**

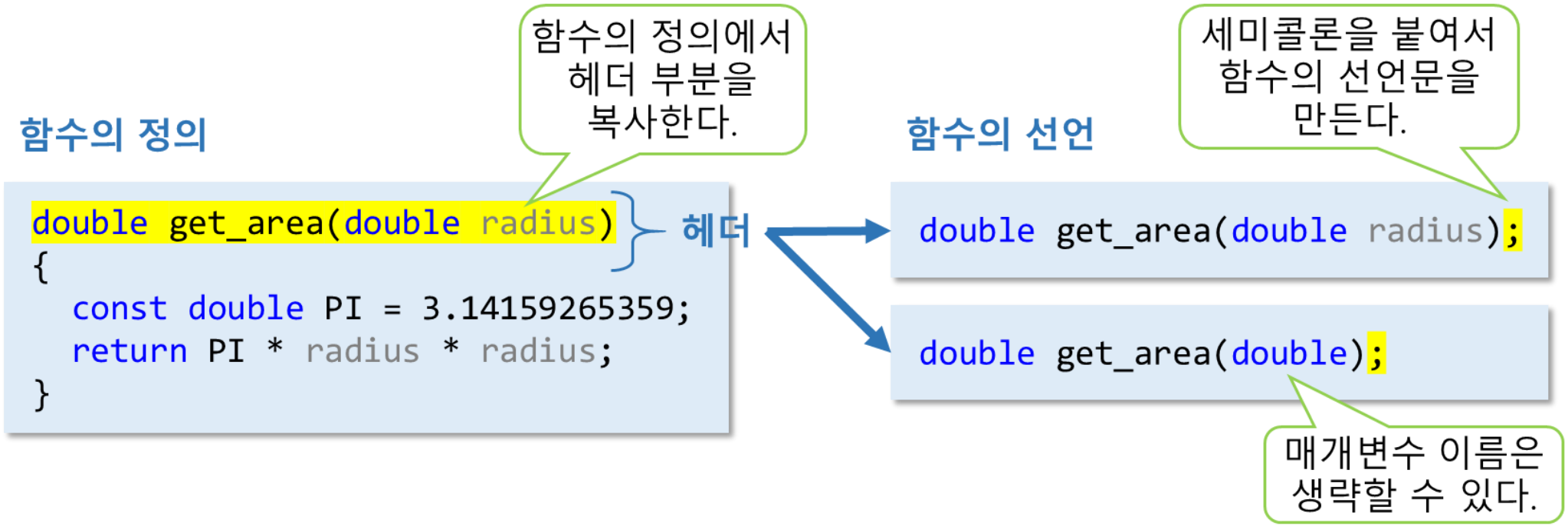


함수 선언문

형식 리턴형 함수명(매개변수목록);

사용예

```
int add(int x, int y);
void hi(void);
void draw_line(char ch, int len);
double get_area(double radius);
```



함수 선언문의 위치

함수 선언문은 #include문 다음에 넣어준다.

```c
#include <stdio.h>

double get_area(double radius);

int main(void)
{
    ⋮
  printf("%f", i, get_area(i));
    ⋮
  return 0;
}

double get_area(double radius)
{
  const double PI = 3.14159265359;
  return PI * radius * radius;
}
```

함수 선언문은 소스 파일의 시작 부분에 넣어준다.

함수가 정의된 위치에 상관 없이 호출할 수 있다.

예제 : draw_line, get_area 함수의 선언

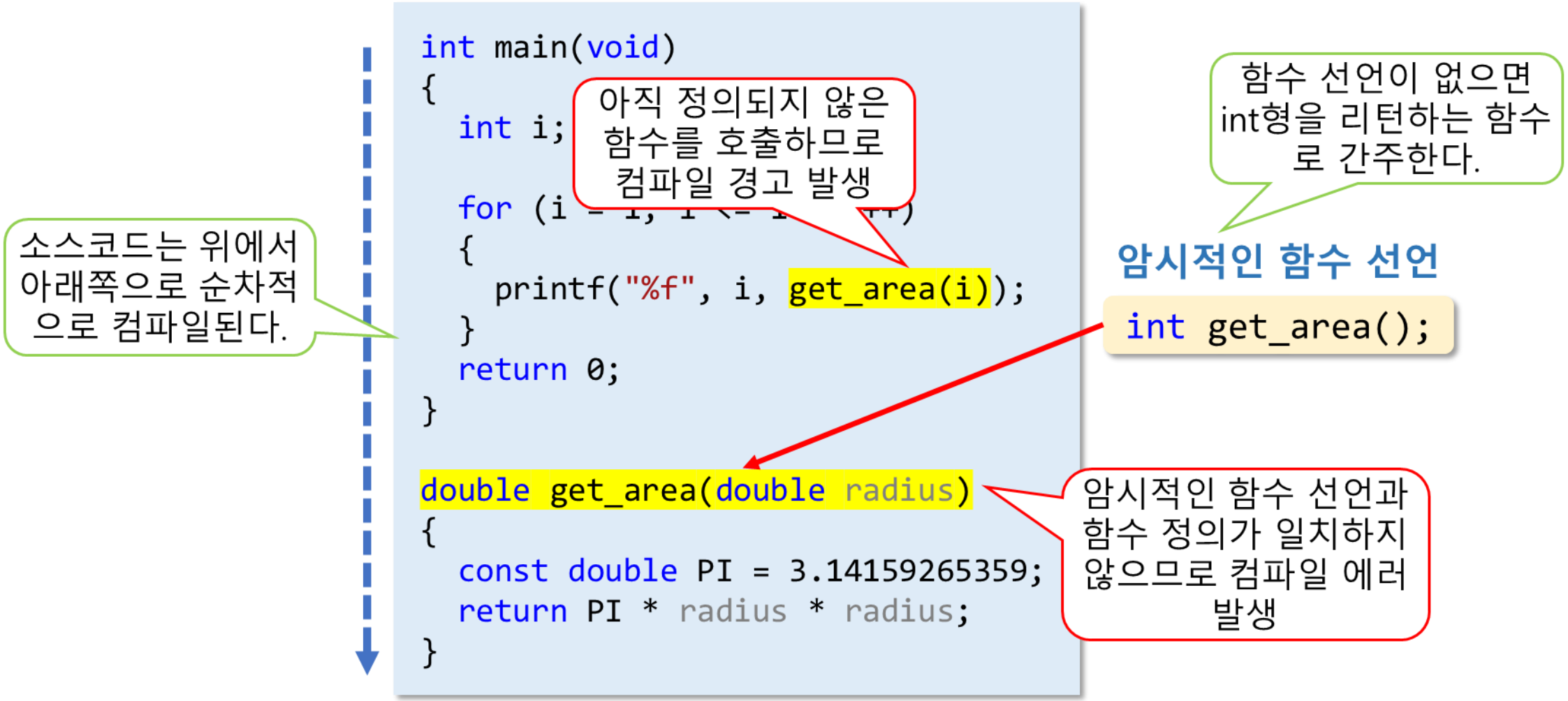
```
03  void draw_line(char ch, int len);
04  double get_area(double radius);
05
06  int main(void)
07  {
08      int i;
09
10      draw_line('-', 40);
11      for (i = 5; i <= 20; i+=5)
12      {
13          printf("반지름이 %d일 때 원의 면적: %.2f\n", i, get_area(i));
14      }
15      draw_line('-', 40);
16      return 0;
17  }
19  double get_area(double radius)          // 원의 면적을 구하는 함수
20  {
21      const double PI = 3.14159265359;
22      return PI * radius * radius;
23  }
24
25  void draw_line(char ch, int len)        // ch를 len개 출력해서 선을 그린다.
26  {
27      int i;
28      for (i = 0; i < len; i++)
29          printf("%c", ch);
30      printf("\n");
31  }
```

함수 선언문

실행결과

```
----------------------------------------
반지름이 5일 때 원의 면적: 78.54
반지름이 10일 때 원의 면적: 314.16
반지름이 15일 때 원의 면적: 706.86
반지름이 20일 때 원의 면적: 1256.64
----------------------------------------
```

선언/정의되지 않은 함수의 호출



- main 함수 앞에 함수 정의를 할경우 선언 필요 없음
  - 단, 많은 함수 정의를 할경우 프로그램 가독성 저해 우려

```
double get_area(double radius)
{
  const double PI = 3.14159265359;
  return PI * radius * radius;
}
```

```
int main(void)
{
    ⋮
  printf("%f", i, get_area(i));
    ⋮
  return 0;
}
```

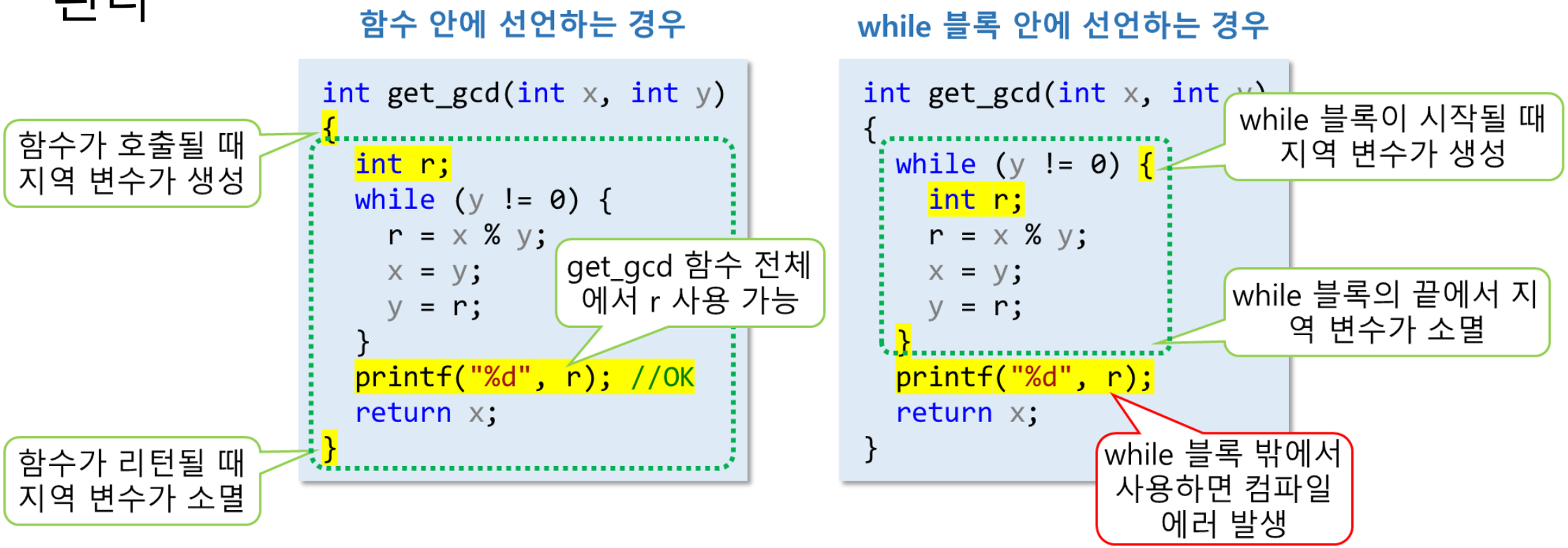
지역 변수와 전역 변수

- **지역 변수(local variable)** : 함수나 블록 안에 선언되는 변수
- **전역 변수(global variable)** : 함수 밖에 선언되는 변수

구분	지역 변수	전역 변수
선언 위치	함수나 블록 안	함수 밖
사용 범위	변수가 선언된 함수나 블록 안	소스 파일 전체
생존 기간	변수가 선언된 블록에 들어갈 때 생성되고 블록을 빠져나갈 때 소멸	프로그램이 시작될 때 생성되고 프로그램이 종료될 때 소멸
초기화하지 않는 경우	쓰레기 값	0으로 초기화

지역 변수

- ANSI C에서는 지역 변수는 블록의 시작부분에 선언해야 한다
- C99에서는 원하는 위치에 지역 변수를 선언할 수 있다
- 지역 변수가 선언된 위치에 따라 지역 변수의 사용 범위가 결정된다



지역 변수의 생성과 소멸

❖함수 안에 선언된 지역 변수
- 함수가 호출되는 횟수만큼 생성되고 소멸된다

```
void dummy()
{
    int y = 456;
    printf("y = %d\n", y);
    y--;
}

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

함수 호출 시 매번 다시 생성

함수 리턴 시 소멸

y = 456을 3번 출력

❖반복문의 블록 안에 선언된 지역 변수
- 반복문이 수행되는 횟수만큼 생성되고 소멸된다

```
for (i = 0; i < 3; i++)
{
    int x = 123;

    printf("x = %d\n", x);

    x++;
}
```

각 반복 회차마다 매번 다시 생성

x = 123을 3번 출력

블록의 끝을 만나면 소멸

예제 : 지역 변수의 생성과 소멸 과정

```
void dummy()
{
    int y = 456;
    printf("y = %d\n", y);
    y--;
}

int main(void)
{
    int i;
    for (i = 0; i < 3; i++)
        dummy();
    for (i = 0; i < 3; i++) {
        int x = 123;
        printf("x = %d\n", x);
        x++;
    }
}
```

함수 호출 시 매번 다시 생성

감소된 y는 함수 리턴 시 소멸

for의 각 반복회차가 시작될 때마다 생성

증가된 x는 블록의 끝을 만나면 소멸

실행결과

```
y = 456
y = 456
y = 456
x = 123
x = 123
x = 123
```

함수의 매개변수

❖함수의 매개변수도 지역 변수다

- 함수가 호출될 때 생성되고 함수가 리턴할 때 소멸된다
- 함수 안에서 함수의 매개변수를 변경해도 함수를 호출하는 쪽에 아무 영향도 주지 않는다

```
void double_it(int num)
{
    num *= 2;
}

int main(void)
{
    int x = 5;
    double_it(x);
    printf("x = %d\n", x);
}
```

매개변수를 변경하는 것은 의미없다

함수 호출 후 x는 변경되지않는다.

x = 5 출력

```
int double_this(int num)
{
    return num * 2;
}

int main(void)
{
    int x = 5;
    x = double_this(x);
    printf("x = %d\n", x);
}
```

함수를 호출하는 쪽으로 전달

함수의 리턴 값을 x에 받아온다.

x = 10 출력

예제 : 지역 변수로서의 매개변수

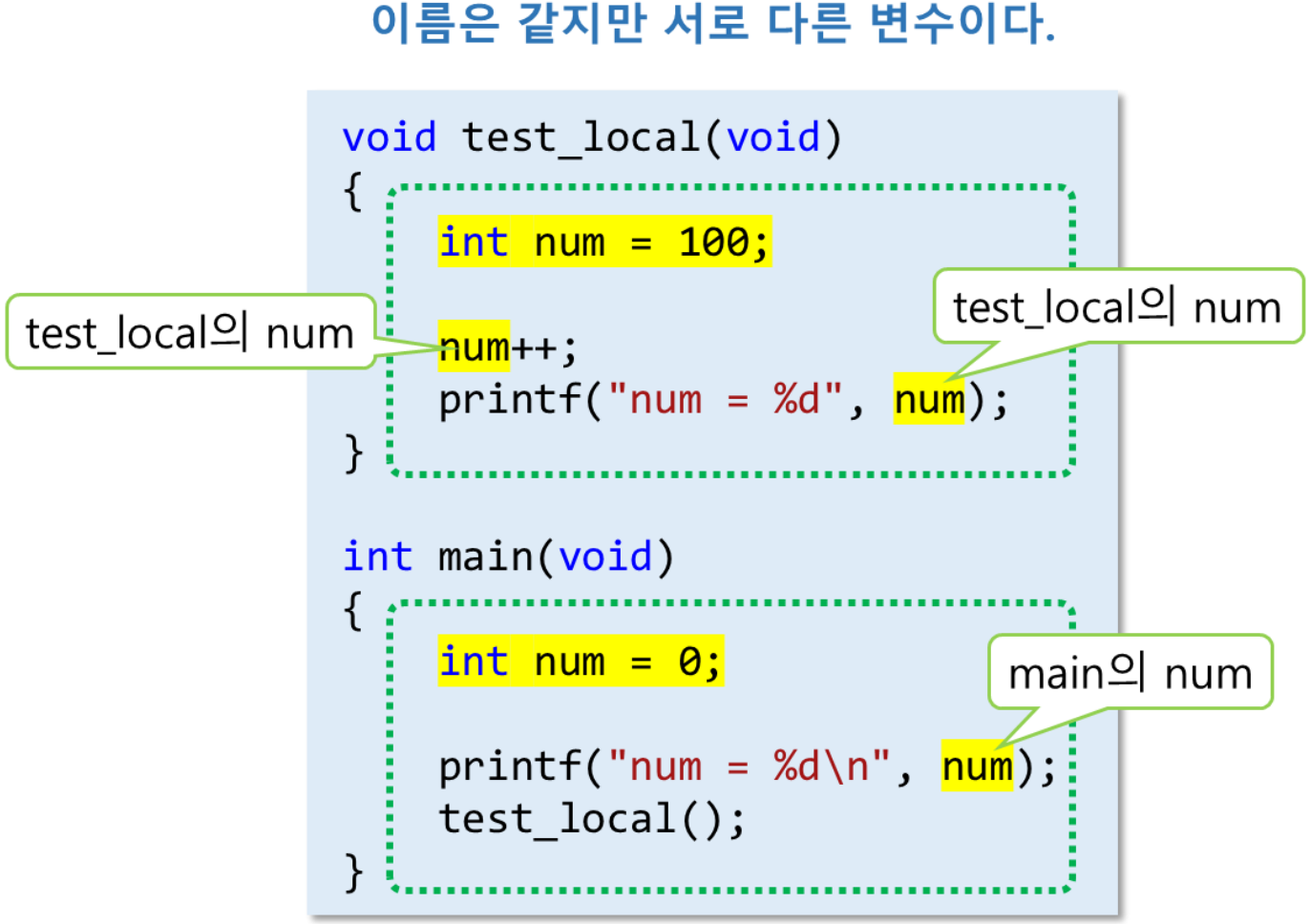
```
03  void double_it(int num)
04  {
05      num *= 2;                   // 매개변수 num은 함수가 리턴할 때 소멸된다.
06  }
07
08  int double_this(int num)
09  {
10      return num * 2;             // 2배로 만든 값을 리턴한다.
11  }
13  int main(void)
14  {
15      int x = 5;
16
17      double_it(x);               // x는 변경되지 않는다.
18      printf("double_it 호출 후 : x = %d\n", x);
19
20      x = double_this(x);         // 함수의 리턴 값을 받아와서 x에 저장한다.
21      printf("double_this 호출 후 : x = %d\n", x);
22
23      return 0;
24  }
```

실행결과

```
double_it 호출 후 : x = 5
double_this 호출 후 : x = 10
```

지역 변수의 사용 범위

- 서로 다른 함수에서 같은 이름의 변수를 선언하면, 이름은 같지만 서로 다른 변수가 된다
- 각각의 함수는 자신 안에 선언된 변수만 사용한다



예제 : 지역 변수의 사용 범위

```
03  void test_local(void);
04
05  int main(void)
06  {
07      int num = 0;    // main의 num 선언
08
09      printf("main: num = %d\n", num);    // main의 num 사용
10      test_local();
11
12      return 0;
13  }
14
15  void test_local(void)
16  {
17      int num = 100; // test_local의 num 선언
18
19      num++;          // test_local의 num 사용
20      printf("test_local: num = %d\n", num);        // test_local의 num 사용
21  }
```

실행결과

```
main: num = 0
test_local: num = 101
```

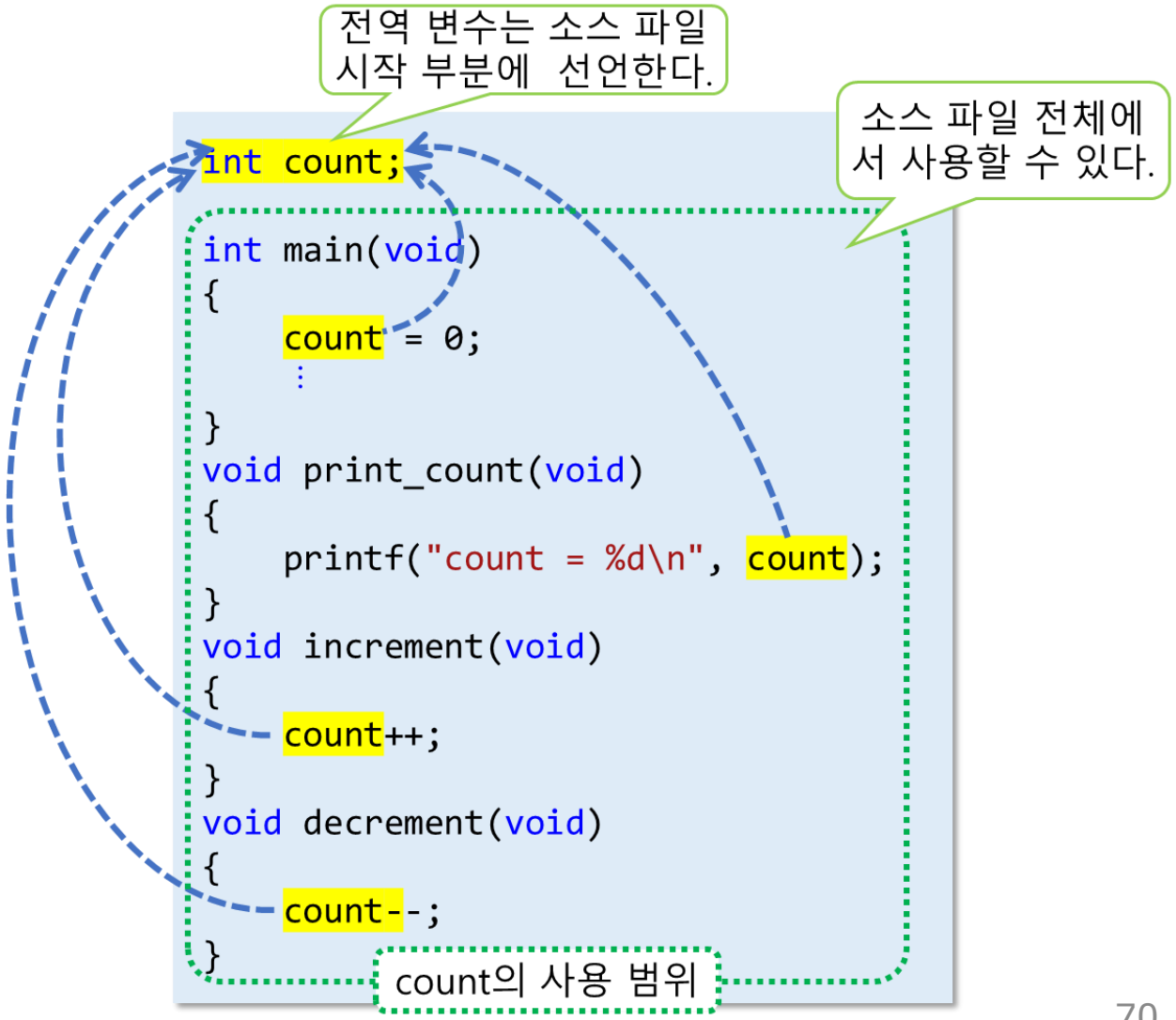
전역 변수의 선언

- 전역 변수는 보통 소스 파일의 시작 부분에 선언한다
- 전역 변수는 따로 초기화하지 않으면 자동으로 0으로 초기화된다

```c
#include <stdio.h>

int count;          // 전역 변수는 선언 시 따로 지정하지 않으면 0으로 초기화된다.

int main(void)
{
    return 0;
}
```

전역 변수

- 전역 변수는 프로그램이 시작될 때 한번만 생성되고, 프로그램이 종료될 때 소멸된다
- 소스 파일의 시작 부분에 전역 변수를 선언하면, 같은 소스 파일의 모든 함수에서 전역 변수를 바로 사용할 수 있다



예제 : 전역 변수의 선언 및 사용

```
void print_count(void);
void increment(void);
void decrement(void);

int count; // 전역 변수 선언

int main(void)
{
  int i;
  count = 0;
  print_count();
  for (i = 0; i < 3; i++)
    increment();
  print_count();
  for (i = 0; i < 3; i++)
    decrement();
  print_count();
}
```

```
void print_count(void) {
  printf("count = %d\n", count);
}
void increment(void) {
  count++;
}
void decrement(void) {
  count--;
}
```

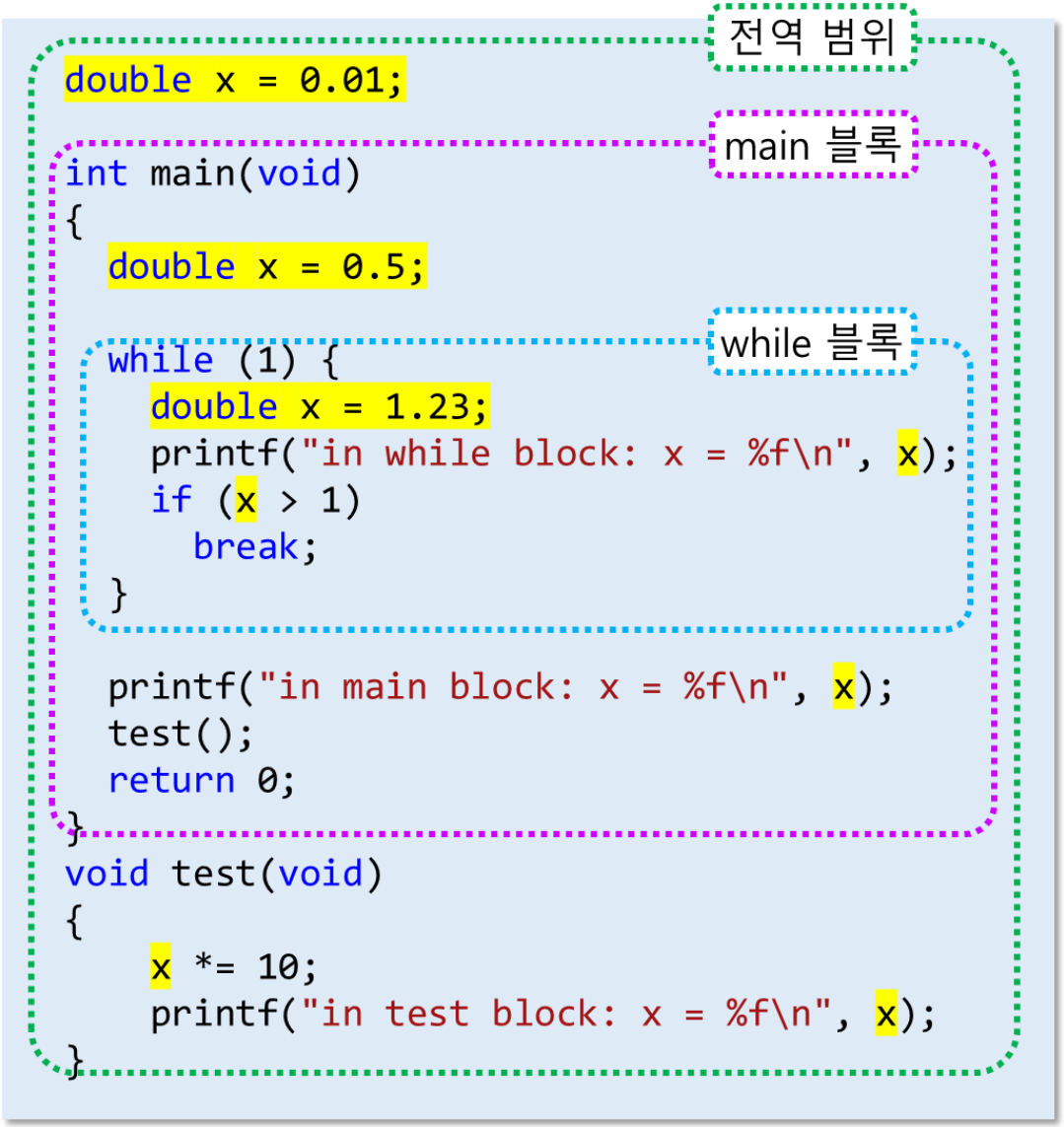
실행결과

```
count = 0
count = 3
count = 0
```

변수의 영역 규칙

- ❖블록 범위가 다를 때는 같은 이름의 변수를 여러 번 선언할 수 있다
- ❖가까운 블록 안 에 선언된 변수가 우선적으로 사용된다
- ❖같은 이름의 지역 변수가 있을 때는 전역 변수를 사용할 수 없다
  - 전역 변수 이름 앞에 g_와 같은 접두사를 사용해서 지역 변수명과 구분한다

```
double g_x = 0.01;
```




```
double x = 0.01;

int main(void)
{
  double x = 0.5;

  while (1) {
    double x = 1.23;
    printf("in while block: x = %f\n", x);
    if (x > 1)
      break;
  }

  printf("in main block: x = %f\n", x);
  test();
  return 0;
}
void test(void)
{
    x *= 10;
    printf("in test block: x = %f\n", x);
}
```

예제 : 변수의 영역 규칙

```
void test(void);
double x = 0.01;  // 전역 변수 x

int main(void) {
  double x = 0.5; // main의 지역 변수 x
  while (1) {
    double x = 1.23;  // while 블록의 지역 변수 x
    printf("in while block: x = %f\n", x);  // x는 1.23
    if (x > 1)        // x는 1.23
      break;
  }
  printf("in main block: x = %f\n", x);     // x는 0.5
  test();
}

void test(void) {
  x *= 10;     // 전역 변수 x
  printf("in test block: x = %f\n", x);
}
```

실행결과

```
in while block: x = 1.230000
in main block: x = 0.500000
in test block: x = 0.100000
```

병렬 블록과 중첩 블록

```
{
        int a, b;
         .....
        {                                   /* inner block 1 */
                float b;
                .....                       /* int a is known, but not int b */
        }
        .....
        {                                   /* inner block 2 */
                float a;
                .....                       /* int b is known, but not int a */
                                            /* nothing in inner block 1 in known */
        }
        .....
}
```

디버깅을 위한 블록 사용

- 블록은 디버깅을 위한 목적으로 많이 사용
- 코드 부분에 임시로 블록을 삽입하면, 프로그램의 다른 부분에 영향을 주지 않는 지역 변수를 사용할 수 있음
- v가 이상한 값을 갖는다고 가정

```
{                       /* debugging starts here */
      static int cnt = 0;
     printf("*** debug : cnt = %d v = %d\n", ++cnt, v);
}
```

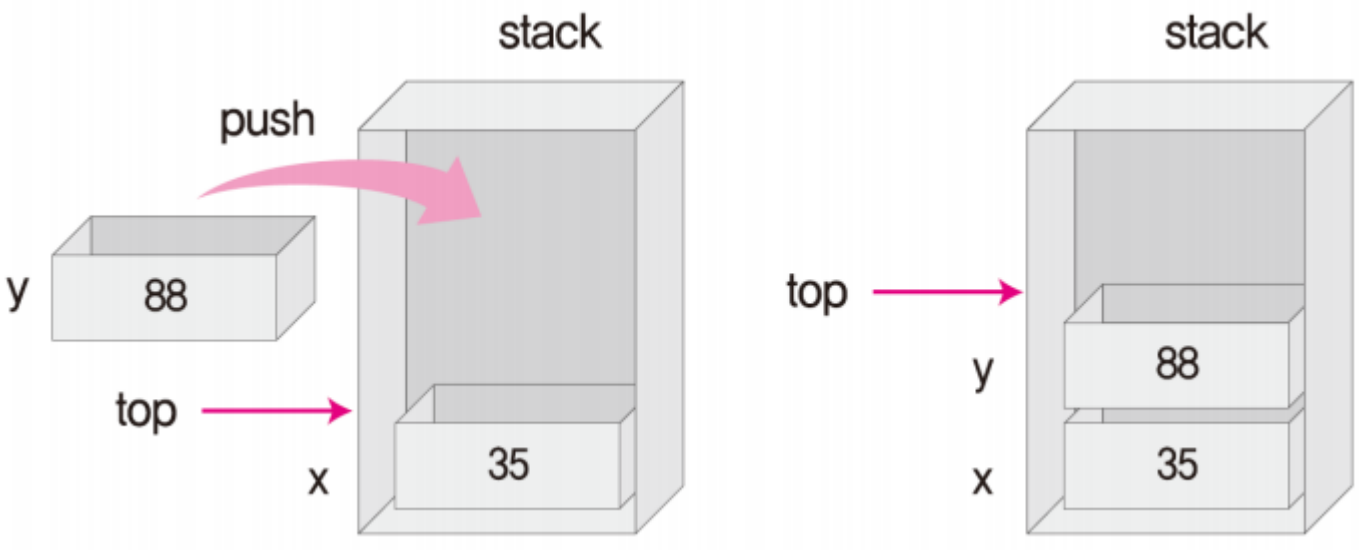
메모리 영역

메모리 영역 분류

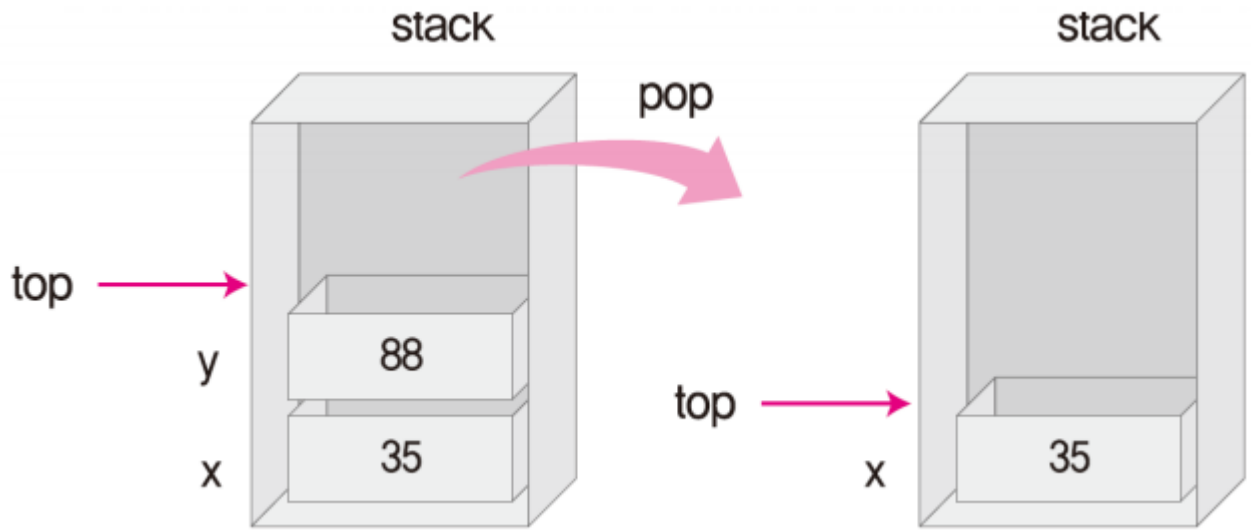
메모리 영역	내용
코드 영역	실행 파일의 바이너리 코드를 저장하는 공간
데이터 영역	• 프로그램이 종료될 때까지 유지할 데이터를 저장할 공간 • 전역변수가 저장
스택 영역	• 함수 내에서만 사용할 데이터(지역변수, 매개변수) 저장 공간 • 블록 내부(일시적)에서만 유효, 블록 안에서 선언
힙 영역	• 동적 할당 메모리 공간(malloc()으로 동적 할당) • 블록 내부(일시적)에서 유효, 블록 안에서(static 사용) 선언 • 프로젝트 내의 여러 파일에서 유효, 함수의 밖에서(extern 사용) 선언 • 하나의 파일(영구적)에서 유효, 함수의 밖에서(static 사용) 선언

메모리 영역

- push 동작



- pop 동작



재귀 함수

- 함수 내부에서 자기 자신의 함수를 다시 호출(재귀 호출)하 는 순환 호출 함수
- 복잡한 알고리즘을 간략하게 구현 가능
- 재귀는 각 호출을 위한 인자와 변수를 스택에 쌓아두어 관 리하기 때문에 많은 시간과 공간을 요구함
- 즉, 재귀를 사용할 때에는 비효율성을 고려해야 함
- 그러나 일반적으로 재귀적 코드는 작성하기 쉽고, 이해하기 쉬우며, 유지보수하기가 쉬움

재귀 함수

❖단순한 재귀적 루틴은 일반적인 패턴을 따름

- 재귀의 일반적인 패턴에서는 기본적인 경우와 일반적인 재귀 경우를 처리하는 코드가 있음
- 보통 두 경우는 한 변수에 의해 결정됨

❖일반적인 재귀 함수의 제어 흐름

1. 변수를 검사하여 기본적인 경우인지 일반적인 경우인지를 결정
2. 기본적인 경우일 때에는 더 이상 재귀 호출을 하지 않고 필요한 값을 리턴
3. 일반적인 경우일 때에는 그 변수의 값이 결국에 기본적인 경우의 값이 될 수 있게 하여 재귀 호출

재귀 함수 예제

```
int sum(int n){
        if (n <= 1)                                    // 1번

                return n;
        else                                           // 2번
                return (n + sum(n - 1));
}
```

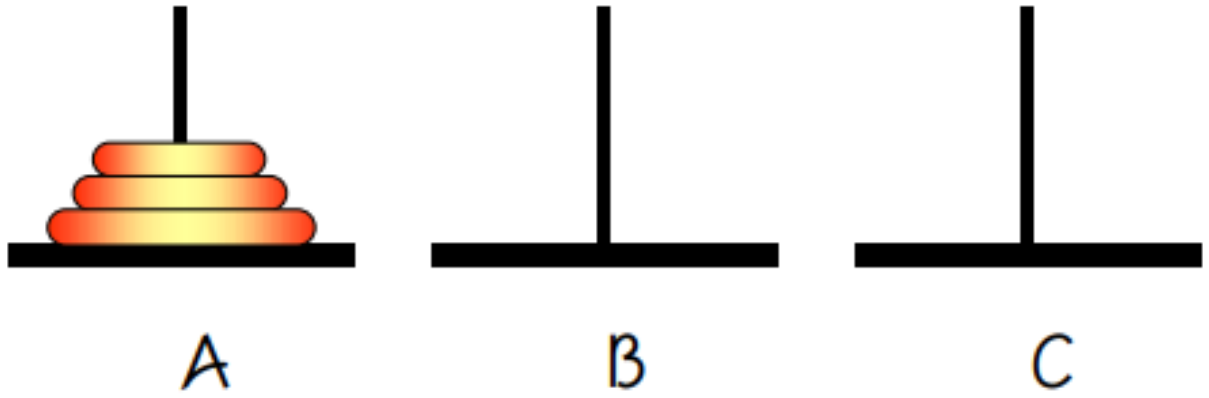
❖위 예제 코드에서는 n을 사용하여 두 경우를 판단

1. n이 1보다 작거나 같으면 기본적인 경우임
   - n을 리턴
2. 아니면 일반적인 경우임
   - n에서 1을 빼어 재귀 호출 ==> n에서 1을 뺐기 때문에 언젠가 n은 1보다 작거나 같아 질 것임

하노이 탑 문제

❖하노이 탑 문제는 막대 A에 쌓여있는 원판 3개를 막대 C로 동일한 모양으로 옮기는 것 *단 다음의 조건을 지켜야 함

- 한 번에 하나의 원판만 이동할 수 있다
- 맨 위에 있는 원판만 이동할 수 있다
- 크기가 작은 원판 위에 큰 원판이 쌓일 수 없다
- 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다



하노이 탑 알고리즘

- 순서대로 1부터 n까지 원판이 있고 A, B, C 3개의 막대기가 있는 경우 하노이 탑 문제를 해결하는 방법

1. A에서 2번부터 n번째 까지 n-1개의 원판을 B로 이동
2. A에서 1번 원판을 C로 이동
3. B에서 2번부터 n번째 까지 n-1개의 원판을 C로 이동

하노이 탑 알고리즘

```
// 막대 from에 쌓여있는 n개의 원판을 막대 tmp를 사용하여 막대 to로 옮긴다.
hanoi_tower(n, 'A', 'B', 'C');                    //하노이 함수 호출


void hanoi_tower(int n, char from, char tmp, char to)
{
        if (n == 1)
                printf("board %d move %c->%c\n", n, from, to);
        else{
                hanoi_tower(n-1, from, to, tmp);
                printf("1board %d move %c->%c\n",n, from, to);
                hanoi_tower(n-1, tmp, from, to);
        }
}
```

표준 C 라이브러리 함수

❖C 컴파일러가 필수로 제공해야하는 플랫폼 독립적인 함수 집합

❖라이브러리 함수를 호출하려면
- 라이브러리 헤더 파일을 포함해야 한다
- 호출할 함수의 원형을 알아야 한다

❖라이브러리 레퍼런스 페이지
- 함수에 대한 대략적인 설명
- 함수 원형
- 매개변수 및 리턴 값에 대한 설명
- 필요한 헤더 파일 및 요구사항
- 간단한 사용 예

헤더 파일	함수	예
stdio.h	표준 입출력에 관한 함수	printf, scanf, puts
conio.h	키보드 및 화면 입출력 함수	getch, cprintf, kbhit
stdlib.h	자료 변환 함수들	atoi, itoa, malloc, free
math.h	수학 함수들	sin, cos, log
string.h	문자열 조작 함수들	strcpy, strlen

<ctype.h>

함수 원형	설명
int isalnum(int c);	알파벳이나 숫자인지 검사한다.
int isalpha(int c);	알파벳인지 검사한다.
int isdigit(int c);	숫자인지 검사한다.
int islower(int c);	소문자인지 검사한다.
int isupper(int c);	대문자인지 검사한다.
int isspace(int c);	공백 문자인지 검사한다.
int isxdigit(int c);	16진수 숫자인지 검사한다.
int tolower(int c);	소문자로 변환한다.
int toupper(int c);	대문자로 변환한다.

<math.h>

함수 원형	설명
double fabs(double x);	절대값을 구한다.
double fmod(double x, double y);	x를 y로 나눈 나머지를 구한다.
double exp(double x);	e^x를 구한다.
double log(double x);	$\log_e x$를 구한다.
double pow(double x, double y);	x의 y승 x^y를 구한다.
double sqrt(double x);	x의 제곱근을 구한다.
double sin(double x);	sin 값을 구한다.
double cos(double x);	cos 값을 구한다.
double tan(double x);	tan 값을 구한다.
double ceil(double x);	x보다 작지 않은 가장 작은 정수를 구한다.
double floor(double x);	x보다 크지 않은 가장 큰 정수를 구한다.

<stdlib.h>

분류	함수 원형	설명
데이터 변환	int atoi(const char *str);	문자열을 정수로 변환한다.
	double atof(const char *str);	문자열을 실수로 변환한다.
	long atol(const char *str);	문자열을 long형 값으로 변환한다.
난수 생성	int rand(void);	난수를 생성한다.
	void srand(unsigned int seed);	난수의 시드를 지정한다.
메모리 관리	void *malloc(size_t size);	size 바이트 크기의 동적 메모리를 할당한다.
	void free(void *memblock);	동적 메모리를 해제한다.
	void *calloc(size_t num, size_t size);	num×size 바이트 크기의 동적 메모리 배열을 할당하고 0으로 초기화한다.
	void *realloc(void *memblock, size_t size);	동적 메모리의 크기를 변경해서 재할당한다.
프로그램 지원 기능	void exit(int exit_code);	프로세스를 exit_code 종료코드로 종료시킨다.
	int system(const char *command);	운영체제가 제공하는 명령을 수행한다.

<string.h>

분류	함수 원형	기능
문자열 처리	char *strcpy(char *dest, const char *src);	문자열을 복사한다.
	char* strncpy(char *dest, const char *src, size_t cnt);	cnt개의 문자열을 복사한다.
	char* strcat(char *dest, const char *src);	문자열을 연결한다.
	char* strncat(char *dest, const char *src, size_t cnt);	cnt개의 문자열을 연결한다.
문자열 검사	size_t strlen(const char *s);	문자열의 길이를 구한다.
	int strcmp(const char *s1, const char *s2);	문자열을 비교한다.
	int strncmp(const char *s1, const char *s2, size_t cnt);	cnt개의 문자열을 비교한다.
	char* strstr(const char *s1, const char *s2);	부분 문자열을 검색한다.
	char* strtok(char *s1, const char *s2);	토큰을 구한다.
메모리 처리	void *memcpy(void *dest, const void *src, size_t cnt);	메모리를 복사한다.
	int memcmp(const void *buf1, const void *buf2, size_t cnt);	메모리를 비교한다.
	void *memmove(void *dest, const void *src, size_t cnt);	메모리를 이동한다.
	void *memset(void *dest, int c, size_t cnt);	메모리를 설정한다.

<time.h>

함수 원형	기능
time_t time(time_t *timer);	시스템 시간을 구한다.
struct tm *localtime(const time_t *timer);	time_t 를 tm 구조체로 변환한다.
time_t mktime(struct tm *timeptr);	tm 구조체를 time_t로 변환한다.
char *asctime(const struct tm *timeptr);	tm 구조체를 문자열로 변환한다.
size_t strftime(char *dest, size_t sz, const char *format, const struct tm *tp);	형식 문자열로 날짜 시간 문자열을 생성한다.
clock_t clock(void);	CPU 클럭수를 리턴한다.

추가 자료

Making coffee 프로그램

```
int main()
{
        int coffee;                                          커피 종류 선택 변수
        printf("어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) ");
        scanf_s("%d", &coffee);                              커피 종류 입력
        printf("\n# 1. 뜨거운 물을 준비한다\n");
        printf("# 2. 종이컵을 준비한다\n");
        switch (coffee)
        {
                case 1 : printf("# 3. 보통커피를 탄다\n"); break;
                case 2 : printf("# 3. 설탕커피를 탄다\n"); break;      안내문 출력
                case 3 : printf("# 3. 블랙커피를 탄다\n"); break;
                default : printf("# 3. 아무거나 탄다\n"); break;
        }
        printf("# 4. 물을 붓는다\n");
        printf("# 5. 스푼으로 저어서 녹인다\n\n");
        printf("손님~ 커피 여기 있습니다.\n\n");
}
```

Making coffee 프로그램 (모듈화)

```
#include <stdio.h>
int coffee_machine(int button)
{
    printf("\n# 1.(자동으로) 뜨거운 물을 준비한다\n");
    printf("# 2. (자동으로) 종이컵을 준비한다\n");
    switch (button)
    {
        case 1 : printf("# 3. (자동으로) 보통커피를 탄다\n");
break;
        case 2 : printf("# 3. (자동으로) 설탕커피를 탄다\n");
break;
        case 3 : printf("# 3. (자동으로) 블랙커피를 탄다\n");
break;
        default : printf("# 3. (자동으로) 아무거나 탄다\n");
break;
    }
    printf("# 4. (자동으로) 물을 붓는다\n");
    printf("# 5. (자동으로) 스푼으로 저어서 녹인다\n\n");
    return 0;
}
```

```
int main()
{
    int coffee;
    int ret;
    printf("어떤 커피를 드릴까요?(1:보통, 2:설탕, 3:블랙)
");
    scanf("%d", &coffee);
    ret = coffee_machine(coffee);
    printf("손님~ 커피 여기 있습니다.\n\n");
}
```

Making coffee 실행결과

실행 결과

어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) 2
1. 뜨거운 물을 준비한다
2. 종이컵을 준비한다
3. 설탕커피를 탄다
4. 물을 붓는다
5. 스푼으로 저어서 녹인다
손님~ 커피 여기 있습니다.

Making coffee 프로그램 – 확장

```
#include <stdio.h>
int coffee_machine(int button)
{
    printf("\n# 1.(자동으로) 뜨거운 물을 준비한다\n");
    printf("# 2. (자동으로) 종이컵을 준비한다\n");
    switch (button)
    {
        case 1 : printf("# 3. (자동으로) 보통커피를 탄다\n"); break;
        case 2 : printf("# 3. (자동으로) 설탕커피를 탄다\n"); break;
        case 3 : printf("# 3. (자동으로) 블랙커피를 탄다\n"); break;
        default : printf("# 3. (자동으로) 아무거나 탄다\n"); break;
    }
    printf("# 4. (자동으로) 물을 붓는다\n");
    printf("# 5. (자동으로) 스푼으로 저어서 녹인다\n\n");
    return 0;
}
```

```
int main()
{
    int coffee;
    int ret;
    printf("A님, 어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) ");
    scanf_s("%d", &coffee);
    ret = coffee_machine(coffee);
    printf("A님 커피 여기 있습니다.\n\n");
    printf("B님, 어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) ");
    scanf ("%d", &coffee);
    ret = coffee_machine(coffee);
    printf("B님 커피 여기 있습니다.\n\n");
    printf("C님, 어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) ");
    scanf_s("%d", &coffee);
    ret = coffee_machine(coffee);
    printf("C님 커피 여기 있습니다.\n\n");
}
```

Making coffee (확장) 실행결과

```
A님, 어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) 1
# 1. (자동으로) 뜨거운 물을 준비한다
# 2. (자동으로) 종이컵을 준비한다
# 3. (자동으로) 보통커피를 탄다
# 4. (자동으로) 물을 붓는다
# 5. (자동으로) 스푼으로 저어서 녹인다
A님 커피 여기 있습니다.

B님, 어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙) 2
# 1. (자동으로) 뜨거운 물을 준비한다
# 2. (자동으로) 종이컵을 준비한다
# 3. (자동으로) 설탕커피를 탄다
# 4. (자동으로) 물을 붓는다
# 5. (자동으로) 스푼으로 저어서 녹인다
B님 커피 여기 있습니다.

C님, 어떤 커피 드릴까요? (1:보통, 2:설탕, 3:블랙)
```

더하기 함수

```
#include <stdio.h>
int plus(int v1, int v2)
{
	int result;
	result = v1 + v2;
	return result;
}
int main()
{
	int hap;
	hap = plus(100, 200);
	printf("100과 200의 plus() 함수 결과는 : %d\n", hap);
}
```

실행 결과

100과 200의 plus() 함수 결과는 : 300

계산기 함수

```
#include <stdio.h>
int calc(int v1, int v2, int op)
{
    int result;
    switch (op )
    {
        case 1 : result = v1 + v2; break;
        case 2 : result = v1 - v2; break;
        case 3 : result = v1 * v2; break;
        case 4 : result = v1 / v2; break;
    }
    return result;
}
```

```
int main()
{
    int res;
    int oper, a, b;
    printf("계산 입력 (1:+, 2:-, 3:*, 4:/) :
");
    scanf("%d", &oper);
    printf("계산할 두 숫자를 입력 : ");
    scanf("%d %d", &a, &b);
    res = calc(a, b, oper);
    printf("계산 결과는 : %d\n", res);
}
```

계산기 함수 실행 결과

실행 결과

```
계산 입력 (1:+, 2:-, 3:*, 4:/) : 3
계산할 두 숫자를 입력 : 7 8
계산 결과는 : 56
```

지역변수와 전역변수

```
#include <stdio.h>
int a = 100;
void func1()
{
        int a = 200;
        printf("func1()에서 a의 값==> %d\n", a);
}
int main()
{
        func1();
        printf("main()에서 a의 값==> %d\n", a);
}
```

실행 결과

```
func1()에서 a의 값==> 200
main()에서 a의 값==> 100
```

함수의 반환값에 따른 차이

```
#include <stdio.h>
void func1()
{
        printf("void 형 함수는 돌려줄게 없음.\n");
}
int func2()
{
        return 100;
}
int main()
{
        int a;
        func1();
        a = func2();
        printf("int 형 함수에서 돌려준 값 ==> %d\n", a);
}
```

실행 결과

void 형 함수는 돌려줄게 없음.
int 형 함수에서 돌려준 값 ==> 100

매개변수 전달 방법 – call by value

```
#include <stdio.h>
void func1(int b)
{
        b = b + 1;
        printf("전달받은 a ==> %d\n", b);
}
void main()
{
        int a=10;
        func1(a);
        printf("func1() 실행 후의 a ==> %d\n", a);
}
```

실행 결과

전달받은 a ==> 11
func1() 실행 후의 a ==> 10

매개변수 전달 방법 비교

```
#include <stdio.h>
void func1 (char a, char b)
{
    int imsi;
    imsi = a;
    a = b;
    b = imsi;
}
void func2 (char *a, char *b)
{
    int imsi;
    imsi = *a;
    *a = *b;
    *b = imsi;
}
```

```
void main()
{
    char x = 'A', y = 'Z';
    printf("원래 값 : x=%c, y=%c\n", x, y);
    func1(x, y);
    printf("값을 전달한 후 : x=%c, y=%c\n", x,
y);
    func2(&x, &y);
    printf("주소를 전달한 후: x=%c, y=%c\n", x,
y);
```

실행 결과

원래 값 : x=A, y=Z
값을 전달한 후 : x=A, y=Z
주소를 전달한 후 : x=Z, y=A

질의 및 응답

참고문헌

- 천정아,『Core C Programming』, 연두에디션(2019)
- C가 보이는 그림책, ANK Co., Ltd. , 성안당 (2018)
- Greg Perry, Dean Miller『어서와 C언어는 처음이지』, 천인국 옮김, 인피니티북스(2015)
- KELLEY (역 : 김명호 외),『A Book on C』, 홍릉과학출판사(2003)
- 윤성우,『 열혈 C 프로그래밍』, 오렌지미디어
- 천인국,『 쉽게 풀어쓴 C언어 Express』, 생능출판사
- 서현우,『 뇌를 자극하는 C 프로그래밍』, 한빛미디어
- 강성수,『 쾌도난마 C프로그래밍』, 북스홀릭
- 고응남,『 C프로그래밍 기초와 응용실습』, 정익사
- 우재남,『 C언어 for Beginner』, 한빛미디어