



C 프로그래밍 및 실습

C Programming

이지민



CHAP 06 함수

학습목표

기본적인 함수의 개념에 대해 알아본다.

함수를 정의할 때 필요한 함수의 리턴형, 함수의 이름, 함수의 매개변수에 대해 알아본다.

함수 호출시 주의 사항과 함수의 선언(또는 원형)에 대해 알아본다.

지역 변수와 전역 변수에 대해 알아본다.

함수의 인자 전달 방법에 대해 알아본다.

목차



함수의 기본

- 함수의 정의
- 함수의 호출
- 함수의 선언



지역변수와 전역변수

- 지역 변수
- 전역 변수
- 변수의 영역규칙

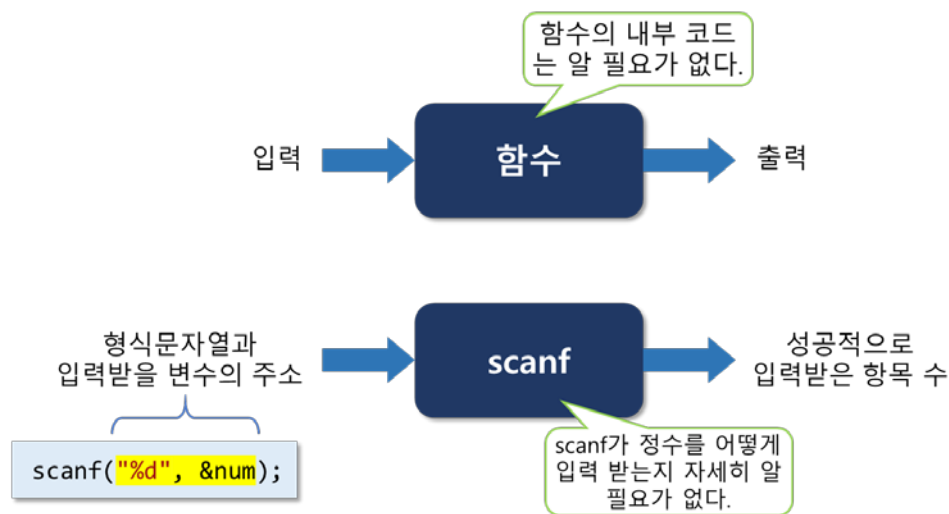
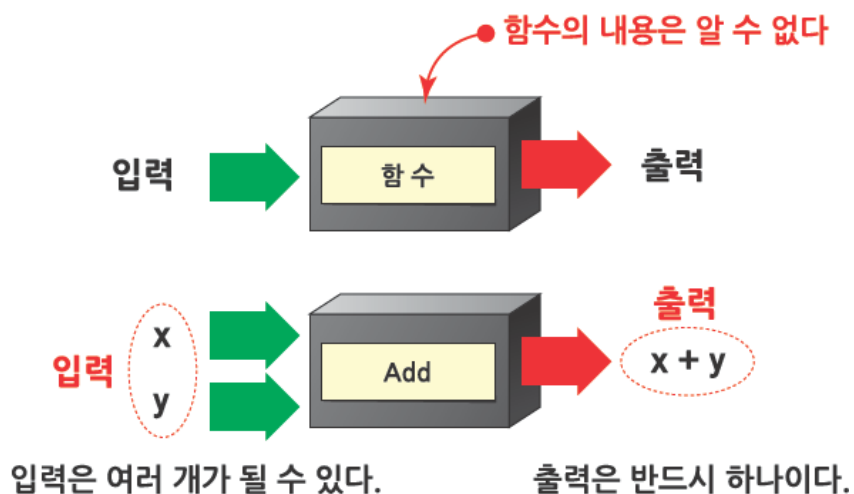


함수의 인자 전달방법

- 값에 의한 전달
- 포인터에 의한 전달

함수의 기본

- ❖ 함수 : 특정 기능을 제공하는 일련의 코드를 묶어서 이름을 붙인 것
- ❖ 일종의 블랙 박스
 - 전달하는 인자의 의미와 호출 결과만 알면 된다.



함수의 기본

❖ 함수를 사용할 때의 장점

- 코드가 중복되지 않으므로 간결하고 알아보기 쉬움
- 한 번 작성해둔 코드를 여러 번 사용하므로 코드의 재사용성이 높음
- 기능 위주로 함수를 작성해서 사용하므로 프로그램의 모듈화 증대
- 함수 코드를 수정하더라도 함수를 호출하는 부분은 수정할 필요가 없으므로 프로그램을 유지 보수하기 쉬움

함수를 사용하지 않는 경우

```
int main(void)
{
    int num, i, sum;
    int result, value;
    scanf("%d", &num);

    sum = 0;
    for ( i = 1 ; i <= num ; i++ )
        sum += i;
    printf("합계 : %d\n", sum);

    ...

    sum = 0;
    for ( i = 1 ; i <= 100 ; i++ )
        sum += i;
    result = sum / 100;

    ...

    sum = 0;
    for ( i = 1 ; i <= value ; i++ )
        sum += i;
    printf("%d까지의 합계 : %d\n", value, sum);

    return 0;
}
```

num까지의 합계를
구하는 코드

100까지의 합계를
구하는 코드

value까지의 합계를
구하는 코드

함수를 사용하는 경우

```
int GetSum(int num)
{
    int i, sum;
    for ( i = 1, sum = 0 ; i <= num ; i++ )
        sum += i;
    return sum;
}

int main(void)
{
    int num, i, sum;
    int result, value;
    scanf("%d", &num);
    sum = GetSum(num);
    printf("합계 : %d\n", sum);

    ...

    result = GetSum(100) / 100;

    ...

    sum = GetSum(value);
    printf("%d까지의 합계 : %d\n", value, sum);

    return 0;
}
```

합계를 구하는 코드는
한 번만 작성

num까지의 합계를
구하는 함수 호출

100까지의 합계를
구하는 함수 호출

value까지의 합계를
구하는 함수 호출

함수의 기본

❖ 함수의 필요성

코드가 길고 복잡해서 알아보기 어렵다.

```
#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.

비슷한 일을 하는 코드가 여러 번 중복된다.

'*'를 24개 출력해서 선 그리는 코드

'-'를 30개 출력해서 선 그리는 코드

```
#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;
}
```

ch를 len개 출력해서 선 그리는 함수

선 그리기 코드는 한 번만 작성하면 된다.

필요할 때마다 함수를 호출한다.

코드가 간결하고 알아보기 쉽다.

함수의 기본

❖ C 프로그램의 함수들

| 종류 | 특징 | 예 |
|-----------|---|--|
| 진입점 함수 | <ul style="list-style-type: none">프로그램이 시작될 때 운영체제에 의해 호출된다.프로그래머가 작성하지만 호출하지는 않는다. | <pre>int main(void) { }</pre> |
| 라이브러리 함수 | <ul style="list-style-type: none">입출력과 같은 고유의 기능을 제공한다.이미 코드가 만들어져 있으므로 프로그래머는 라이브러리 헤더를 포함하고 호출하면 된다. | <pre>#include <stdio.h> char ch; scanf("%c", &ch); printf("%c", ch);</pre> |
| 사용자 정의 함수 | <ul style="list-style-type: none">프로그래머가 직접 정의하고 호출하는 함수이다.프로그램에서 특정 기능을 제공하는 코드 부분을 묶어서 함수로 만들어두고 사용한다. | <pre>int add(int x, int y) { return x + y; } printf("%d", add(10, 20));</pre> |

함수의 기본

❖ 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. | <pre>double get_area(double radius) { : }</pre> |
| 함수의 호출 | 앞에서 선언되거나 정의된 함수를 이용한다. 인자를 넘겨주고 리턴 값을 받아올 수 있다. | <pre>printf("%f", get_area(i));</pre> |
| 함수의 선언 | 함수 호출에 필요한 리턴형, 함수 이름, 매개변수 정보를 알려준다. | <pre>double get_area(double radius);</pre> |

함수의 기본_함수의 정의

❖ 함수를 정의하는 기본적인 형식

- 함수를 정의할 때는 리턴형과 함수 이름, 매개변수 목록이 필요하다.
 - 매개변수 목록은 데이터형과 매개변수가 한 쌍이며, 하나 이상인 경우 콤마 (,)로 나열한다.
 - 매개변수가 없을 때는 void라고 적거나 비워둔다.
 - 함수가 처리할 내용은 { } 안에 적어준다.

형식

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

사용예

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

함수의 기본_함수의 정의

❖ 함수의 리턴형

- 함수가 처리 결과로 리턴하는 값의 데이터 형

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

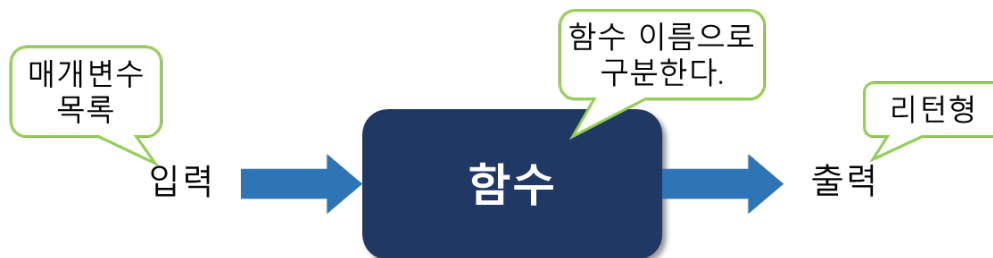
- 함수의 리턴형이 void형이면 함수의 리턴 값이 없다는 의미

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

- 함수의 리턴형을 생략하면 int형으로 간주된다.

```
draw_line(char ch, int len) { ... } // int draw_line(char ch, int len)의 의미
```

- 함수는 반드시 하나의 값만 리턴할 수 있다.
 - 함수의 처리 결과가 둘 이상일 때는 매개변수를 이용해야 한다.



함수의 기본_함수의 정의

❖ 함수의 이름

- 함수의 이름도 식별자를 만드는 규칙에 따라서 만들어야 한다.

| | | |
|---|---------------------------------|-----------------------|
| <code>int get_factorial(int num) { ... }</code> | <code>// 팩토리얼을 구하는 함수</code> | } 소문자로 된 함수 이름 |
| <code>void open_file(void) { ... }</code> | <code>// 파일 열기를 수행하는 함수</code> | |
| <code>void read_data(void) { ... }</code> | <code>// 데이터 읽기를 수행하는 함수</code> | |
| | | |
| <code>int GetFactorial(int num) { ... }</code> | <code>// 팩토리얼을 구하는 함수</code> | } 대소문자를 혼용하는 함수 이름 |
| <code>void OpenFile(void) { ... }</code> | <code>// 파일 열기를 수행하는 함수</code> | |
| <code>void ReadData(void) { ... }</code> | <code>// 데이터 읽기를 수행하는 함수</code> | |

- 어떤 일을 하는 함수인지 명확하게 알 수 있는 이름을 선택한다.

| | |
|--|---|
| <code>void f1(void) { ... }</code> | <code>// 어떤 일을 하는 함수인지 알 수 없다.</code> |
| <code>void play_video(void) { ... }</code> | <code>// 동영상을 재생하는 함수라는 것을 알 수 있다.</code> |

- 일관성 있는 이름을 사용하는 것이 좋다.

함수의 기본_함수의 정의

❖ 함수의 이름

- 이름이 같은 함수를 여러 번 정의할 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.

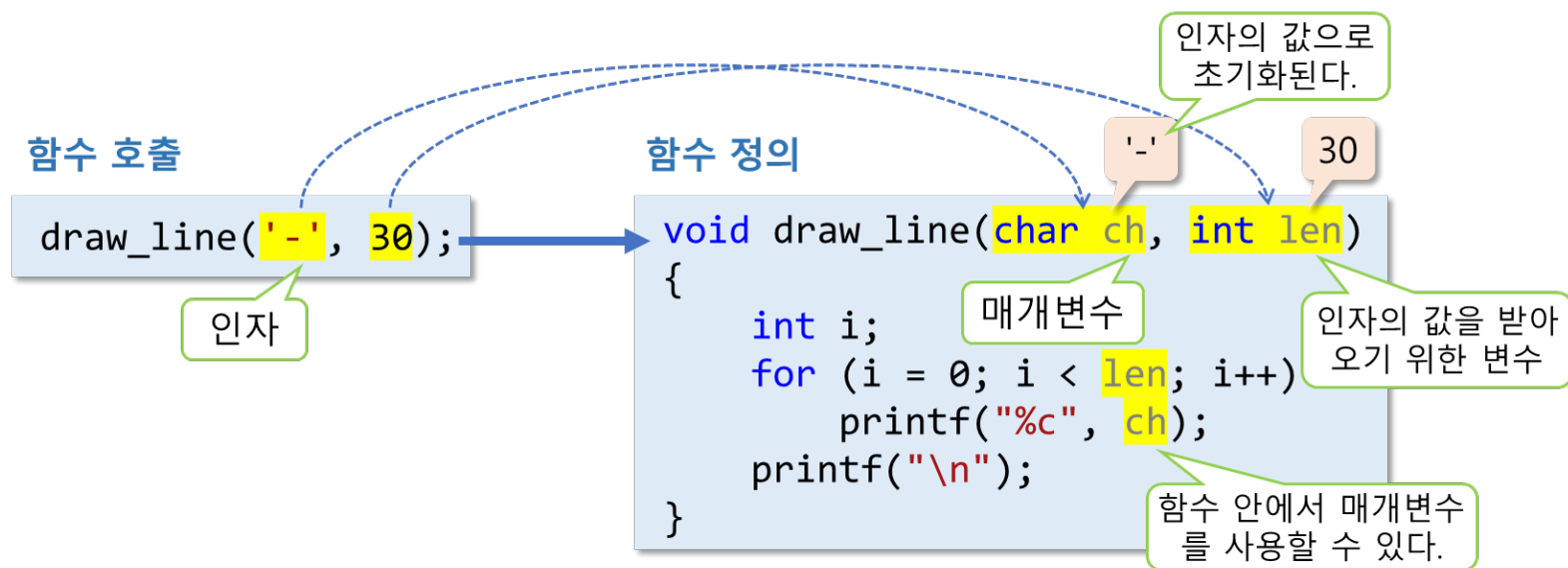
❖ 명명 규칙(Naming Convention)

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

함수의 기본_함수의 정의

❖ 매개변수(parameter)

- 함수를 호출한 곳에서 함수 안으로 전달되는 값을 보관하기 위한 변수
- 함수를 호출하는 쪽과 함수 내부를 연결한다.
- 매개변수는 함수가 호출될 때 생성되고 함수 안에서 사용된다.
 - 함수를 호출 시 넘겨주는 인자의 값으로 초기화



❖ 인자, 인수(argument)

- 함수를 호출할 때, 실제로 전달되는 값(실인자, 실인수라고도 함)

함수의 기본_함수의 정의

❖ 함수의 매개변수는 개수에 제한이 없다.

- 매개변수가 없을 때는 void를 써주는데, 이때의 void는 생략할 수 있다.
- 매개변수가 하나 이상일 때는 콤마(,)로 나열한다.
- 매개변수의 개수에는 제한이 없다

```
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 기본_함수의 정의

❖ 함수의 헤더와 바디

- 함수의 정의는 헤더와 바디로 구성된다.
- 헤더 : 함수의 리턴형, 함수의 이름, 매개변수 목록을 적는 부분
- 바디 : { } 안에 함수가 처리할 문장을 나열한 부분
 - 함수의 바디도 블록이다.

함수를 호출하는데
필요한 정보를 제공
하는 부분

헤더

```
void draw_line(char ch, int len)
```

함수가 처리할 내용
을 적어주는 부분

바디

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

함수의 기본_함수의 정의

❖ 매개변수와 리턴 값이 모두 없는 함수

- 리턴형과 매개변수 목록을 void로 지정한다.
- void형의 함수를 호출하면 따로 return문을 쓰지 않아도 함수의 끝을 만나면 자동으로 리턴한다.

```
void hi(void)
```

```
{
```

```
    printf("Hi! Let's enjoy C programming.\n");
```

```
    return;
```

```
    printf("never executed");
```

```
}
```

리턴형이 void인 함수는 리턴값 없이 리턴한다.

return 이후의 문장은 수행되지 않는다.


```
void bye() { printf("Bye!\n"); }
```

간단한 함수는 한 줄로 작성할 수 있다.

매개변수 목록의 void는 생략할 수 있다.

함수의 기본_함수의 정의

❖ 매개변수는 갖지만 리턴 값이 없는 함수(1/2)

```
void PrintSumAndAverage(int a, int b)  매개변수는 갖지만 리턴 값이 없는 함수
{
    printf("합계 : %d\n", a + b);
    printf("평균 : %f\n", (double) (a + b) / 2);
}
```

선 그리는 기능

```
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");
}
```

매개변수는 함수 안에서 마음대로 사용할 수 있다.

함수의 기본_함수의 정의

❖ 매개변수는 갖지만 리턴 값이 없는 함수(2/2)

count 개의 정수를
입력받아 합계를 구
해서 출력하는 함수

```
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.
}
```

함수의 기본_함수의 정의

- ❖ 매개변수와 리턴 값을 모두 갖는 함수(1/2)
 - 리턴형이 있는 함수에서 return문을 생략하면 안된다.

```
int GetSum(int num)  ◯———— 매개변수와 리턴 값을 갖는 함수
{
    int i;
    int sum = 0;
    for(i = 1 ; i <= num ; i++)
        sum += i;
    return sum;  ◯———— 반드시 int형의 값을 리턴해야 한다.
}
```

```
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.
}
```

num 팩토리얼
을 구하는 함수

리턴형과 같은 형
의 값을 리턴한다.

함수의 기본_함수의 정의

❖ 매개변수와 리턴 값을 모두 갖는 함수(2/2)

```
double GetMax(double a, double b, double c) — 매개변수와 리턴 값을 갖는 함수
{
    double max;
    max = a > b ? a : b;
    max = c > max ? c : max;
    return max; — 반드시 double형의 값을 리턴해야 한다.
}
```

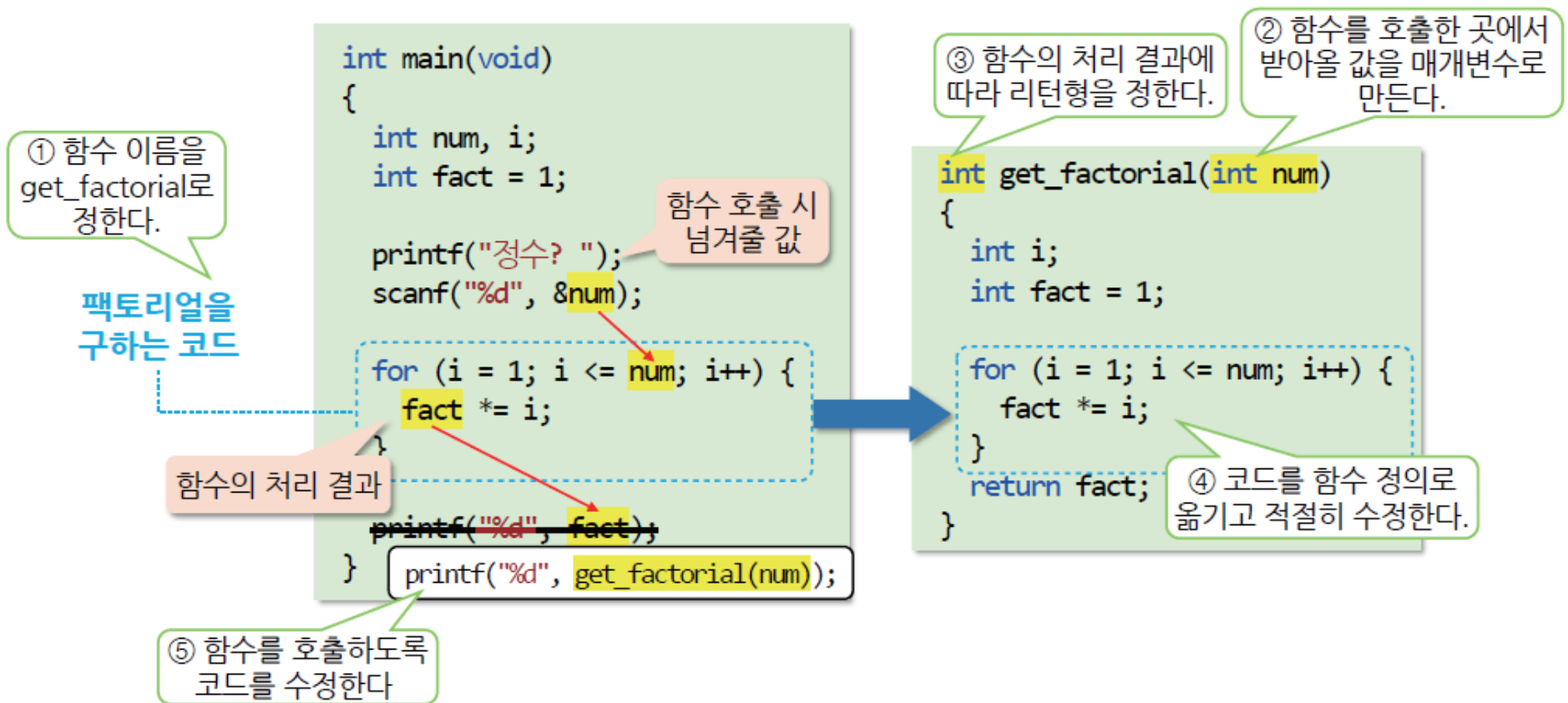
```
double get_area(double radius)
{
    const double PI = 3.14159265359;
    return PI * radius * radius;
}
```

원의 면적을
구하는 함수

매개변수와 리턴값을 모두
double형으로 지정한다.

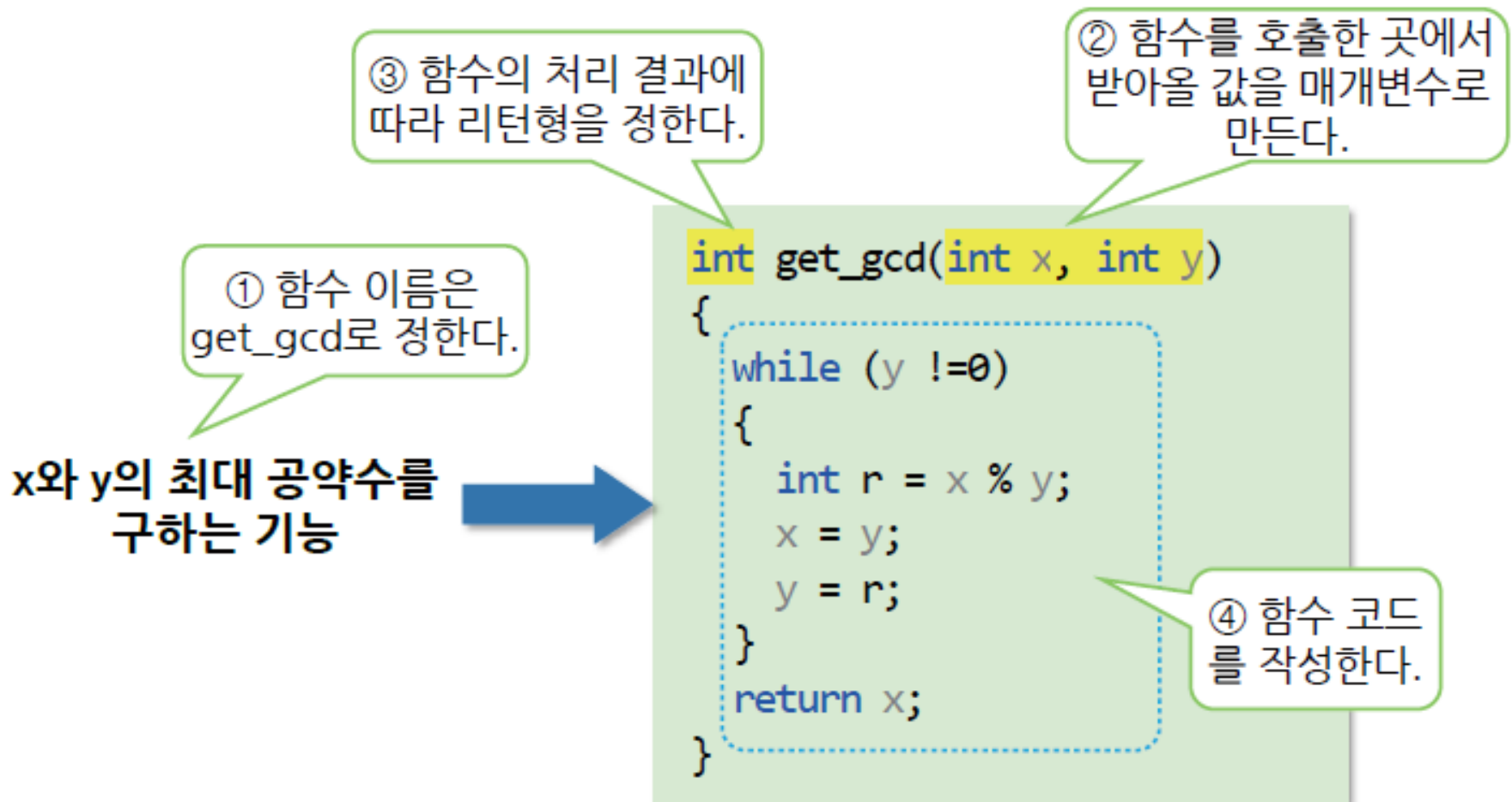
함수의 기본_함수의 정의

❖ 기존 코드에서 함수 정의하기



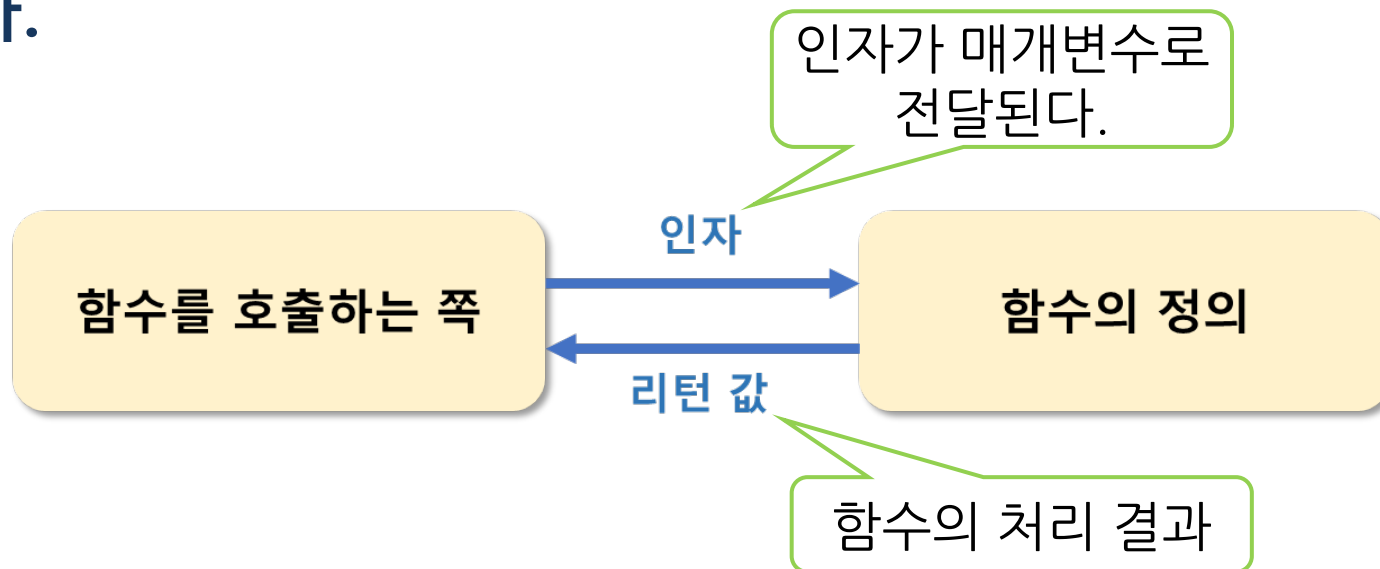
함수의 기본_함수의 정의

❖ 처음부터 함수 정의하기



함수의 기본_함수의 호출

- ❖ 이미 만들어진 함수를 불러 쓰는 것
- ❖ 함수 정의는 한번만 작성하고 필요한 만큼 여러 번 호출할 수 있다.
- ❖ 함수를 호출할 때는 함수의 이름을 쓰고 () 안에 함수의 **인자**를 써준다.
- ❖ 함수를 호출하는 쪽과 함수의 정의는, 서로 데이터를 주고받는다.



함수의 기본_함수의 호출

❖리턴 값과 매개변수가 없는 함수의 호출

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

hi();

hi;

()가 없으면 함수 호출이 아니다.

② 함수의 코드를 수행한다.

① 함수 정의로 이동한다.

③ 함수의 끝을 만나면 호출한 곳으로 돌아온다.

④ 그 다음 코드를 계속 수행한다.


```
void hi(void)
{
    printf("Hi!\n");
}
```

```
int main(void)
{
    hi();
    bye();
    :
    return 0;
}
```

함수의 기본_함수의 호출

❖리턴 값과 매개변수가 없는 함수의 호출 예(1/2)

```
01: /* Ex06_01.c */
02: #include <stdio.h>
03:
04: void PrintHello(void)
05: {
06:     printf("Hello World\n");
07: }
08:
09: void PrintBye(void)
10: {
11:     printf("프로그램을 종료합니다.\n");
12: }
13:
14: int main(void)
15: {
16:     PrintHello( );
17:     PrintBye( );
18:
19:     return 0;
20: }
```



PrintHello 함수의 정의

PrintBye 함수의 정의

실행 결과

프로그램을 시작합니다.
프로그램을 종료합니다.

함수의 기본_함수의 호출

❖리턴 값과 매개변수가 없는 함수의 호출 예(2/2)

```
01  #include <stdio.h>
02
03  void hi(void)
04  {
05      printf("Hi! Let's enjoy C programming.\n");
06  }
```

```
08  void bye() { printf("Bye!\n"); }
```

```
11  int main(void)
```

```
12  {
```

```
13      hi();
```

```
14      bye();
```

```
16      hi();
```

```
17      bye();
```

```
18  }
```

실행 결과

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

간단한 함수는 한 줄로 작성할 수 있다.

매개변수가 없으므로 ()만 써준다.

같은 함수를 여러 번 호출할 수 있다.

함수의 기본_함수의 호출

❖ 매개변수가 없는 함수의 호출 과정

- ① 함수 호출문을 만나면 지금까지 수행하던 문장의 위치를 보관해 두고 함수 정의로 이동한다.
- ② 함수 정의 안에 있는 코드를 순서대로 수행한다.
- ③ return문이나 함수의 끝을 만나면, 함수를 호출한 곳으로 돌아온다.
- ④ 함수 호출문의 다음 문장을 계속해서 수행한다.

함수의 기본_함수의 호출

❖리턴 값은 없고 매개변수만 있는 함수의 호출

- 인자를 갖는 함수를 호출할 때는 ()안에 함수의 인자를 콤마(,)로 나열한다.
- 함수 호출문에 지정된 인자는 순서대로 함수의 매개변수로 전달된다.
- 매개변수의 데이터형과 같은 형의 인자를 사용해야 한다.
 - 인자와 매개변수의 데이터형이 일치하지 않으면 암시적인 형 변환이 일어난다.

```
draw_line('-', 24);
```

```
draw_line(101, 30);
```

101을 char형으로 변환
해서 인자로 사용한다.

함수의 기본_함수의 호출

❖리턴 값은 없고, 두 개의 매개변수를 갖는 함수의 사용 예(1/2)

```
01: /* Ex06_02.c */
02: #include <stdio.h>
03:
04: void PrintSumAndAverage(int a, int b)
05: {
06:     printf("합계 : %d\n", a + b);
07:     printf("평균 : %f\n", (double) (a + b) / 2);
08: }
09:
10: int main(void)
11: {
12:     int x, y;
13:
14:     PrintSumAndAverage(10, 20);
15:
16:     printf("정수를 입력하세요 : ");
17:     scanf("%d %d", &x, &y);
18:     PrintSumAndAverage(x, y);
19:
20:     return 0;
21: }
```

함수의 정의

인자를 갖는 함수의 호출

인자를 갖는 함수의 호출

실행 결과

합계 : 30
평균 : 15.000000
두 정수를 입력하세요 : 15 20
합계 : 35
평균 : 17.500000

함수의 기본_함수의 호출

❖리턴 값은 없고, 매개변수가 있는 함수의 사용 예(2/2)

```
01  #include <stdio.h>
02
03  void draw_line(char ch, int len)
04  {
05      int i;
06      for (i = 0; i < len; i++)
07          printf("%c", ch);
08      printf("\n");
09  }
10
11  int main(void)
12  {
13      int amount = 10, price = 1000;
14      int total = 0, width = 0;
```

*ch*를 *len*개만큼 출력해서
선을 그리는 함수

함수의 기본_함수의 호출

❖리턴 값은 없고, 매개변수가 있는 함수의 사용 예(2/2)

```
16 draw_line('-', 24);
17
18 printf("수량   단가   합계\n");
19 width = 3 + 8 + 8 + 2;
20 draw_line('=', width);
21 total = amount * price;
22 printf("%3d %8d %8d\n", amount, price, total);
23
24 draw_line('-', 24);
25 }
```

'-'를 24개만큼 출력해서 선을 그린다.

헤더 폭을 계산한다.

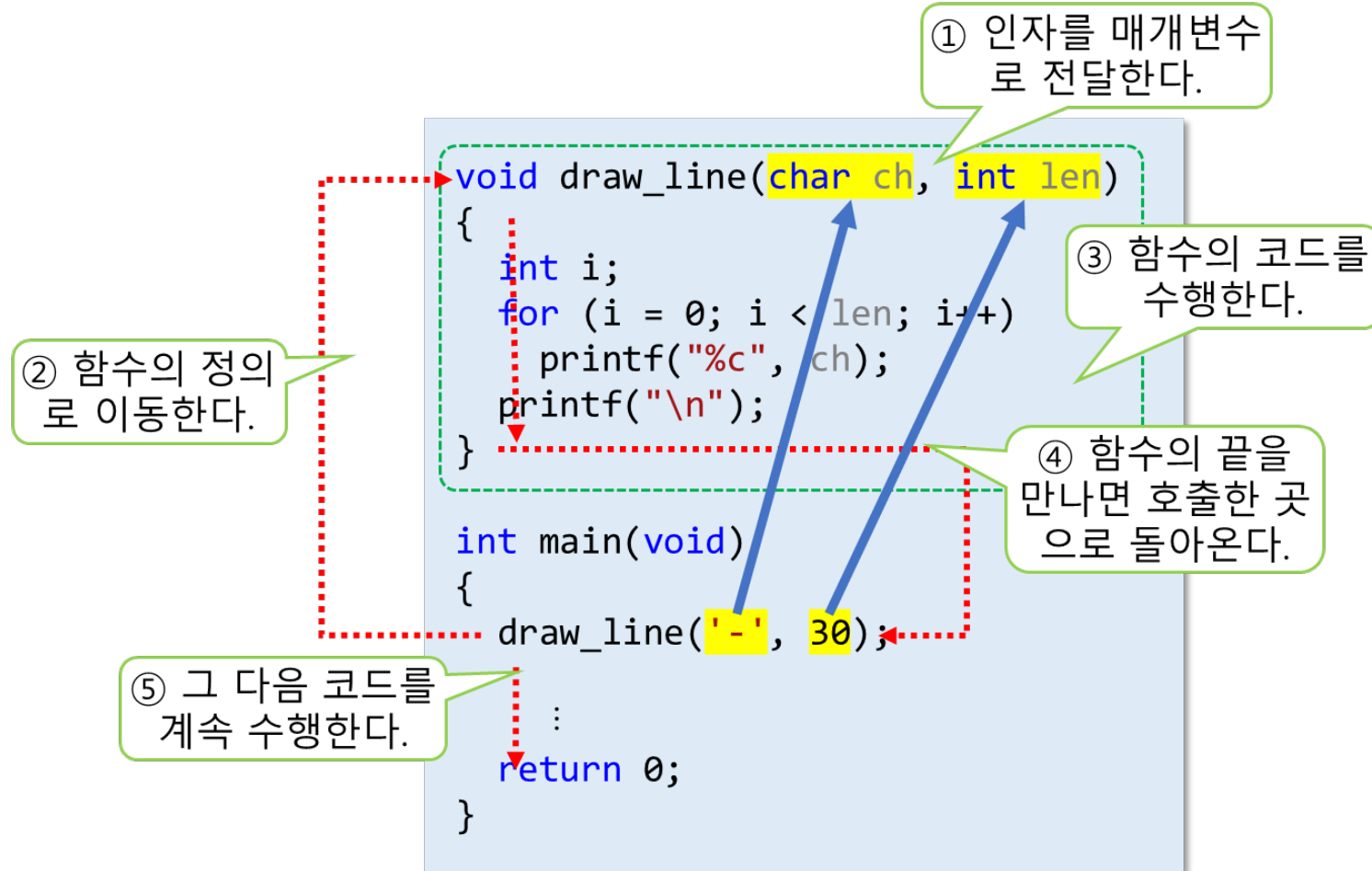
'='를 width개만큼 출력해서 선을 그린다.

실행 결과

| 수량 | 단가 | 합계 |
|----|------|-------|
| 10 | 1000 | 10000 |

함수의 기본_함수의 호출

❖ 매개변수가 있는 함수의 호출 과정



함수의 기본_함수의 호출

❖ 매개변수가 있는 함수의 호출 시 주의 사항(1/2)

- 항상 인자와 매개변수의 개수가 같아야 한다.

❌ `draw_line('-');` // 인자와 매개변수의 개수가 다르므로 컴파일 에러

- 인자와 매개변수의 순서도 같아야 한다.

- 순서가 바뀌어도 컴파일 에러가 발생하지 않으므로 매개변수의 의미에 맞게 순서대로 인자를 전달하도록 주의해야 한다.

`draw_line(24, '-');`

24 문자를 '-'개 출력한다는 뜻

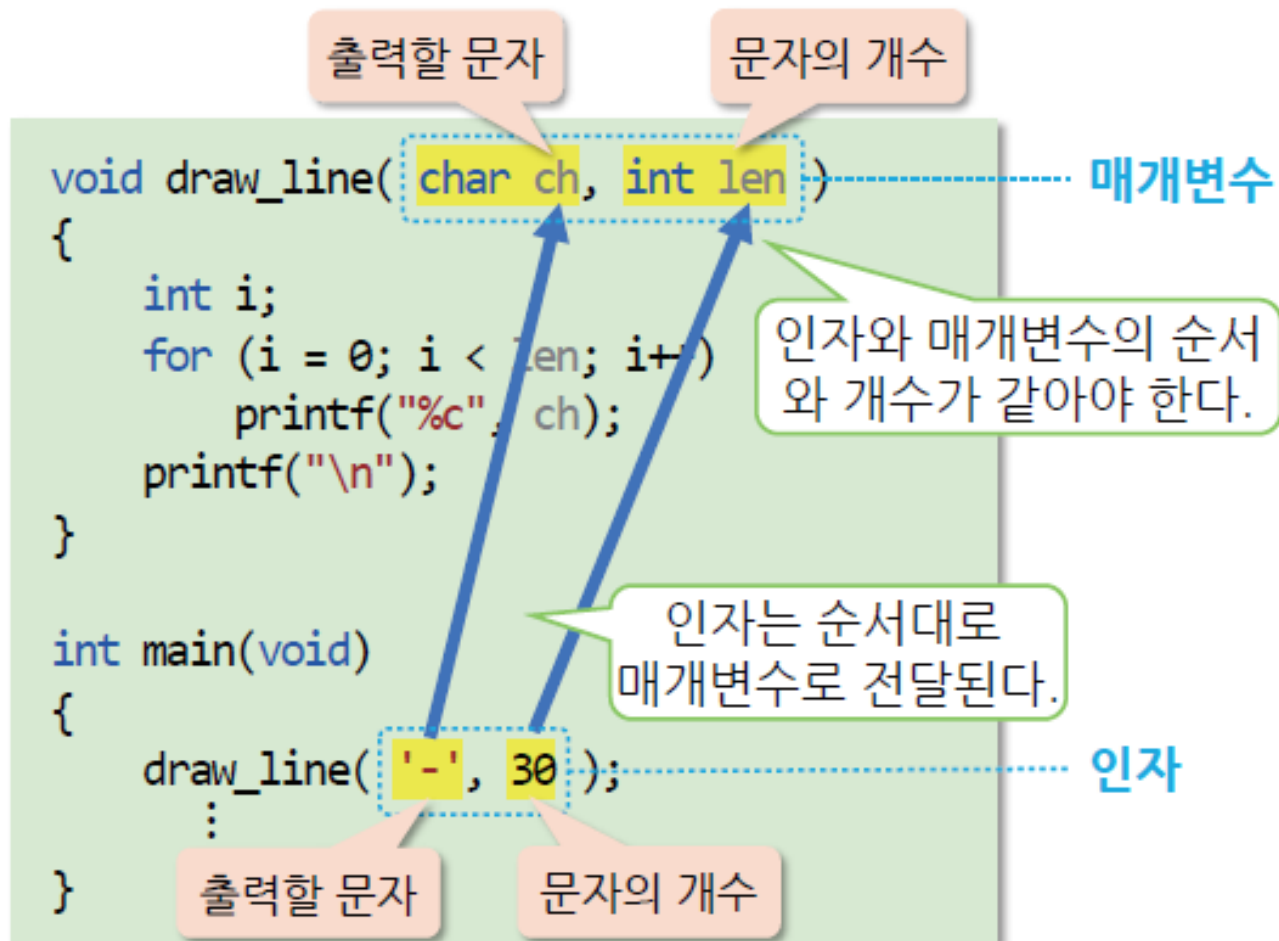
- 리턴형이 void형인 함수는 리턴값이 없으므로 수식이 아니다.

❌ `result = draw_line('-', 24);`

리턴형이 void인 함수 호출문은 수식으로 사용할 수 없다.

함수의 기본_함수의 호출

❖ 매개변수가 있는 함수의 호출 시 주의 사항(2/2)



함수의 기본_함수의 호출

❖ 리턴값과 매개변수가 있는 함수의 호출

- 함수의 리턴값을 다른 수식에 이용할 수 있다. → 리턴값이 있는 함수의 호출문도 수식이다.

```
result = get_factorial(3);
```

리턴값을 변수에 대입할 수 있다.

```
if (get_factorial(num) > 100)  
    printf("big enough");
```

리턴값을 피연산자로 사용할 수 있다.

```
printf("%d", get_factorial(5));
```

리턴값을 함수의 인자로 사용할 수 있다.

- 함수의 리턴값을 사용하지 않을 수도 있다.

```
get_factorial(5);
```

함수의 리턴값도 임시값이므로 사용하지 않으면 사라진다.

함수의 기본_함수의 호출

❖리턴값과 매개변수가 있는 함수의 호출 예(1/2)

```
01  #include <stdio.h>
02
03  int get_factorial(int num)
04  {
05      int i;
06      int result = 1;
07
08      for (i = 1; i <= num; i++)
09          result *= i;
10      return result;
11  }
12
```

팩토리얼값 num!을 구해서
리턴한다.

함수의 기본_함수의 호출

❖리턴값과 매개변수가 있는 함수의 호출 예(2/2)

```
13  int main(void)
14  {
15      int i;
16      int fact;
17
18      for (i = 0; i <= 5; i++) {
19          fact = get_factorial(i);
20          printf("%2d! = %3d\n", i, fact);
21      }
22      get_factorial(5);
23  }
```

get_factorial(i)의 리턴
값을 fact에 대입한다.

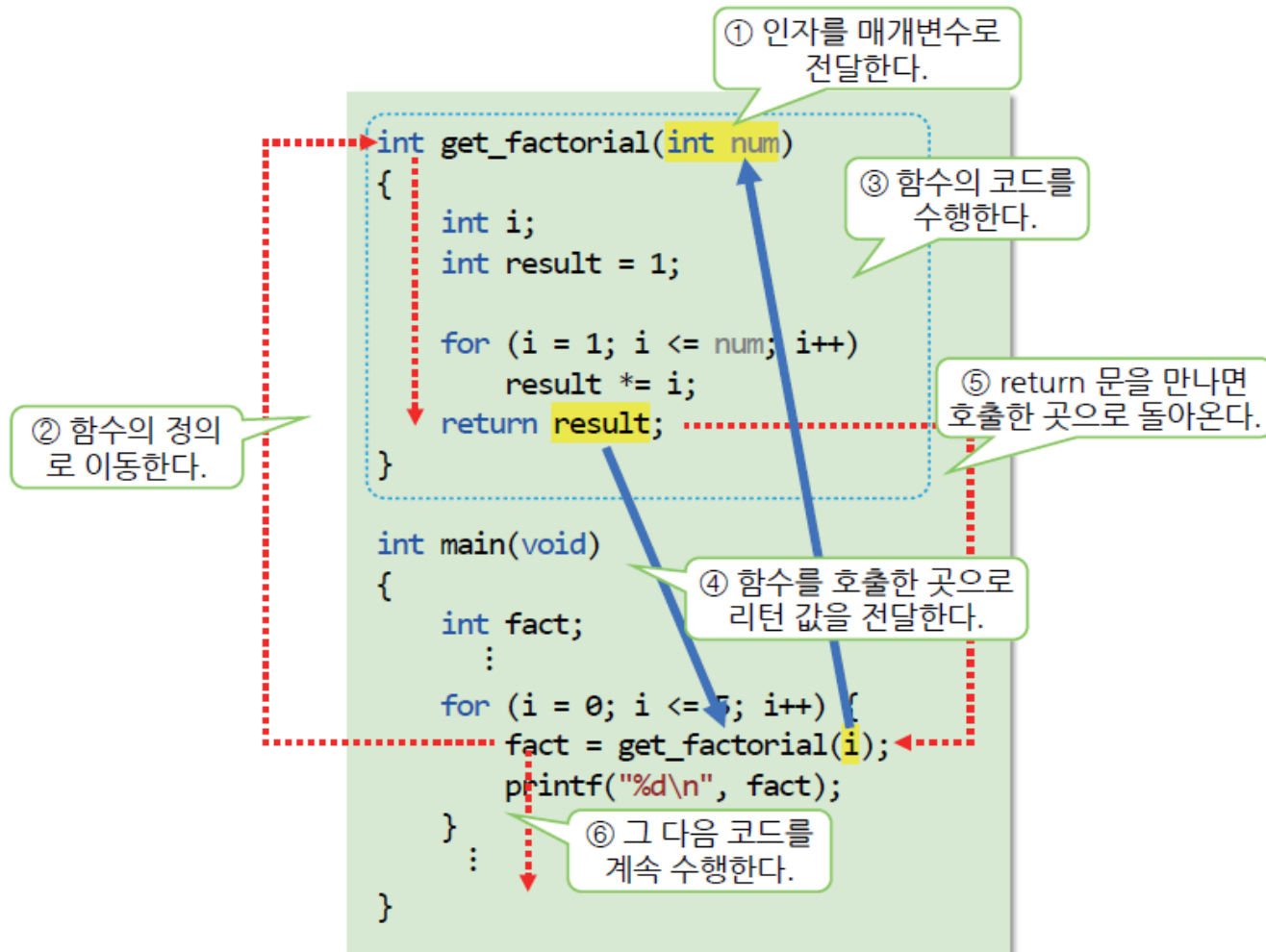
함수의 리턴값을 사용하지
않을 수도 있다.

실행 결과

| | | |
|----|---|-----|
| 0! | = | 1 |
| 1! | = | 1 |
| 2! | = | 2 |
| 3! | = | 6 |
| 4! | = | 24 |
| 5! | = | 120 |

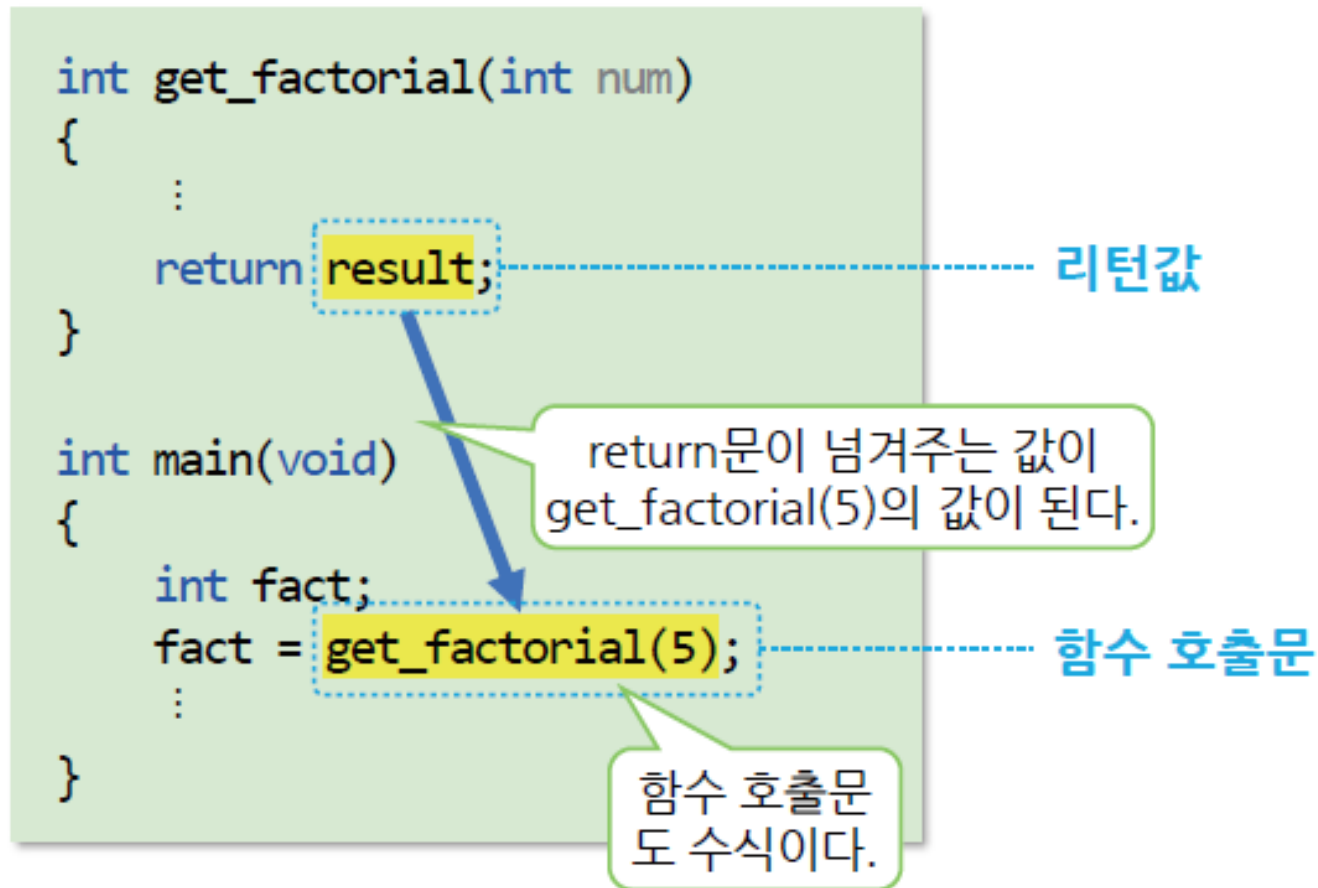
함수의 기본_함수의 호출

❖리턴값과 매개변수가 있는 함수의 호출 과정



함수의 기본_함수의 호출

- ❖ 리턴값은 함수 안에서 함수를 호출한 곳으로 전달된다.
 - return문이 넘겨주는 값이 함수 호출문의 값이 된다.



함수의 기본_함수의 호출

❖리턴값을 전달하는 함수의 호출 예

```
01  #include <stdio.h>
03  double get_area(double radius)
04  {
05      const double pi = 3.14;
06      return pi * radius * radius;
07  }
09  int main(void)
10  {
11      int r;
12      for (r = 1; r <= 5; r++) {
13          printf("r=%d, 원의 면적= %.2f\n", r, get_area(r));
14      }
15  }
```

원의 면적을 구해서 리턴한다.

r을 double형으로 변환해서 전달한다.

실행 결과

r=1, 원의 면적= 3.14
r=2, 원의 면적= 12.56
r=3, 원의 면적= 28.26
r=4, 원의 면적= 50.24
r=5, 원의 면적= 78.50

함수의 기본_함수의 호출

❖ 인자 전달과정의 의미

- 매개변수는 함수가 호출될 때 생성되며, 인자의 값으로 초기화된다.
 - 인자와 매개변수의 데이터형이 같지 않으면, 인자를 매개변수의 데이터형으로 형 변환해서 전달한다.

```
double get_area(double radius)
{
    const double pi = 3.14;
    return pi * radius * radius;
}

int main(void)
{
    int r;
    for (r = 1; r <= 5; r++) {
        printf("%f\n", get_area(r));
    }
}
```

인자를 매개변수의 데이터형에 맞춰 형 변환한다.

매개변수가 생성될 때 인자로 초기화된다.

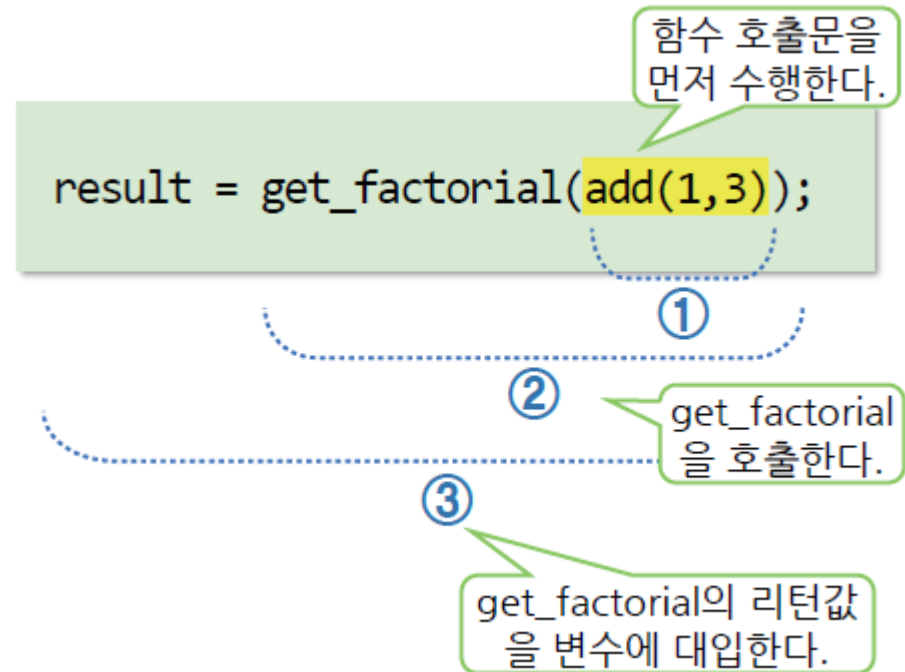
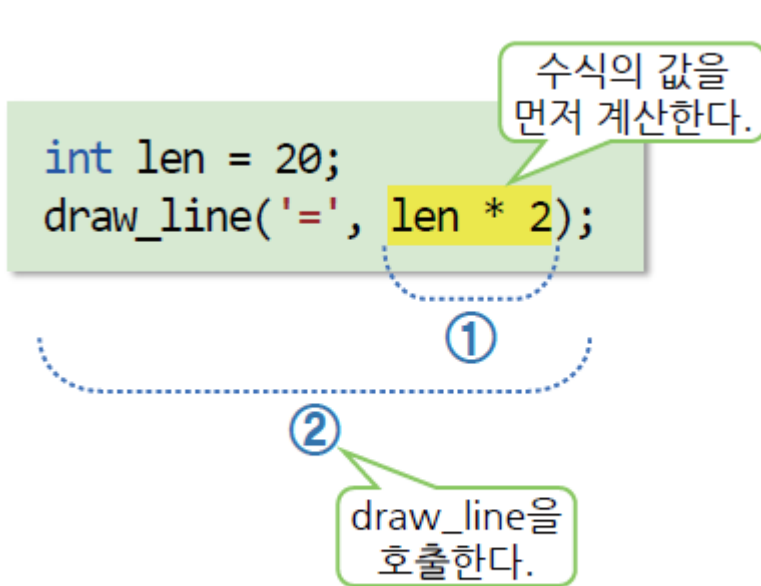
인자가 매개변수로 전달될 때 실제로 수행되는 문장

`double radius = r;`

함수의 기본_함수의 호출

❖ 함수 호출 시 주의사항(1/3)

- 함수의 인자도 수식이다.
 - 인자의 값을 먼저 평가한 다음 함수 호출을 수행한다.



함수의 기본_함수의 호출

❖ 함수 호출 시 주의사항(2/3)

- 함수를 호출할 때 넘겨주는 인자의 개수는 함수 정의에 있는 매개변수의 개수와 일치해야 한다.

매개변수가 3개인 함수

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

get_max(10, 20);

인자가 매개변수보다 부족
하면 컴파일 에러 발생

get_max(10, 20, 30, 40);

인자가 매개변수보다 많으면
컴파일 경고 발생

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

암시적인 형 변환

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

인자를 매개변수의 데이터
형으로 형 변환한다.

12가 리턴된다.

데이터의 손실 발생

컴파일 경고

함수의 기본_함수의 호출

❖ 함수 호출 시 주의사항(3/3)

- 함수를 호출할 때 인자는 반드시 넘겨주어야 하지만 함수의 리턴 값은 받아올 수도 있고, 받아오지 않을 수도 있다.

```
int result;
```

```
result = PrintSumAndAverage(10, 20);
```

PrintSumAndAverage는 리턴 값이 없으므로 다른 수식에서 사용할 수 없다.

```
GetMax(x, y, z);
```

함수의 리턴 값을 다른 수식에 사용하지 않아도 된다.

- 함수는 이름으로 구분한다.
 - 대소문자까지 정확히 일치하는 이름으로 함수를 호출해야 한다.

❌ `get_ma(1, 2, 3);` // 잘못된 이름으로 호출하면 링크 에러

함수의 기본_함수의 호출

❖ 매개변수 개수와 인자 개수에 따른 함수의 호출 예(1/2)

```
01  #include <stdio.h>
02
03  int get_max(int x, int y, int z)
04  {
05      int max = x > y ? x : y;
06      max = z > max ? z : max;
07      return max;
08  }
09
10  int main(void)
11  {
```

매개변수가 3개인 함수

실행 결과

max = 12.000000

함수의 기본_함수의 호출

❖ 매개변수 개수와 인자 개수에 따른 함수의 호출 예(2/2)

```
12 //get_max(10, 20);
13 //get_max(10, 20, 30, 40);
14
15 double max;
16 max = get_max(12.34, 0.5, 7.9);
17 printf("max = %f\n", max);
18
19 //get_ma(1, 2, 3);
20 }
```

인자의 개수가 부족하면
컴파일 에러 발생

인자의 개수가 많으면
컴파일 경고 발생

인자를 매개변수의 데이터형으로
형 변환해서 호출한다.
max = get_max(12, 0, 7);

잘못된 이름으로 호출하면
링크 에러 발생

함수의 기본_함수의 선언

❖ 함수를 호출하려면 먼저 함수가 정의되어 있어야 한다.

소스 코드는 위에서 아래쪽으로 순차적으로 컴파일된다.

```
double get_area(double radius)
{
    const double pi = 3.14;
    return pi * radius * radius;
}
```

```
int main(void)
{
    printf("%f\n", get_area(1.5));
}
```

앞에서 정의된 함수를 호출할 수 있다.

함수 호출문보다 앞쪽에
함수를 정의하는 경우

```
int main(void)
{
    printf("%f\n", get_area(1.5));
}
```

```
double get_area(double radius)
{
    const double pi = 3.14;
    return pi * radius * radius;
}
```

아직 정의되지 않은
함수를 호출하므로
컴파일 경고 발생

함수 호출문보다 뒤쪽에
함수를 정의하는 경우

함수의 기본_함수의 선언

❖ 함수의 정의보다 앞쪽에서 함수를 호출하는 예(1/2)

```
01: #include <stdio.h>
02:
03: int main(void)
04: {
05:     int i_res;
06:     double f_res;
07:
08:     i_res = GetFactorial(5);
09:     printf("5! = %d\\n", i_res);
10:
11:     f_res = GetMax(0.5, 10.5, 12.5);
12:     printf("최대값 = %f\\n", f_res);
13:
14:     return 0;
15: }
16:
```

← 아직 정의되지 않은 함수의 호출
(컴파일 경고)

← 아직 정의되지 않은 함수의 호출
(컴파일 경고)

함수의 기본_함수의 선언

❖ 함수의 정의보다 앞쪽에서 함수를 호출하는 예(2/2)

```
17: int GetFactorial(int num)
18: {
19:     int i;
20:     int fact = 1;
21:     for(i = 1 ; i <= num ; i++)
22:         fact *= i;
23:     return fact;
24: }
```

← GetFactorial 함수의 정의

```
25:
26: double GetMax(double a, double b, double c)
27: {
28:     double max;
29:     max = a > b ? a : b;
30:     max = c > max ? c : max;
31:     return max;
32: }
```

← GetMax 함수의 정의

실행 결과

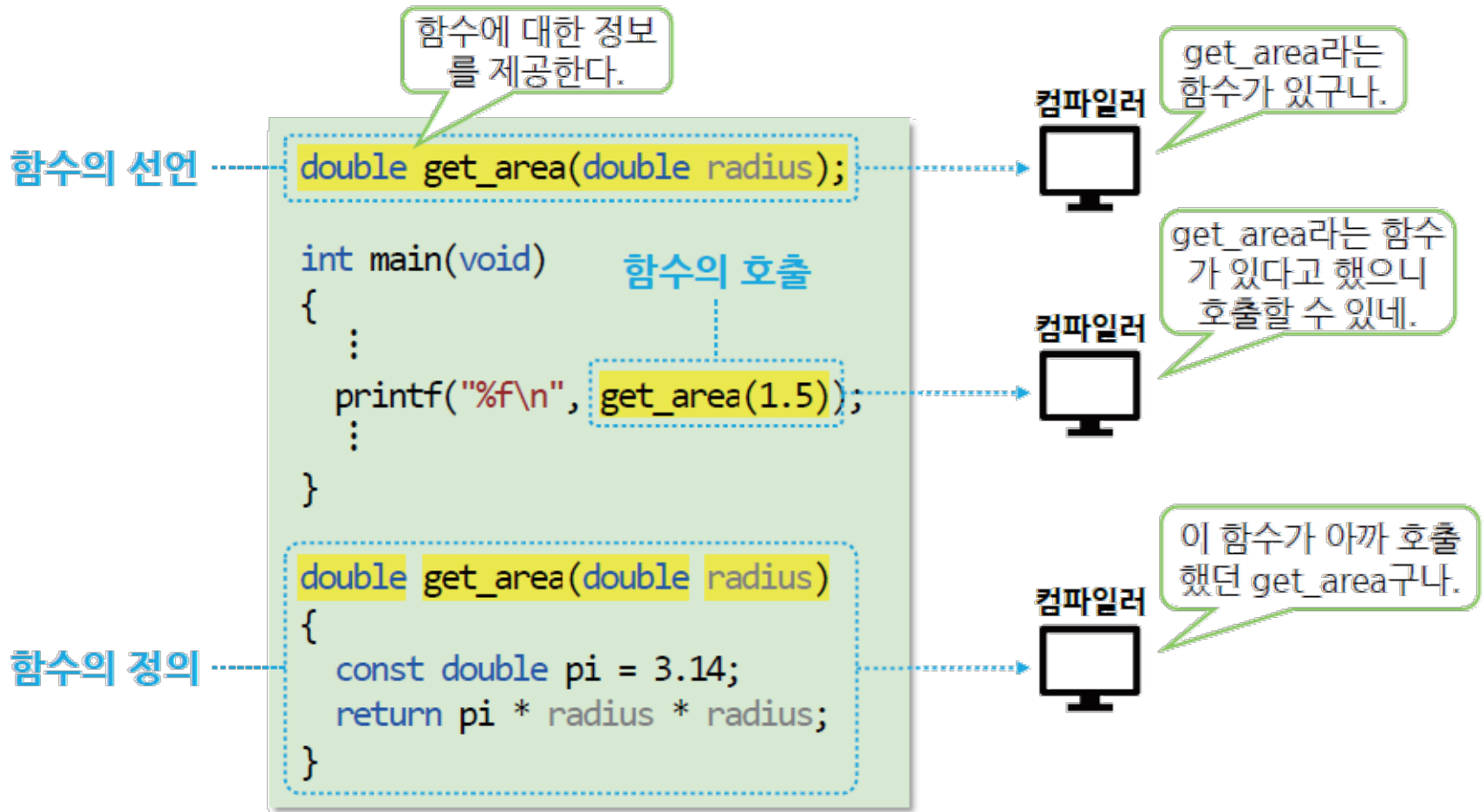
5! = 120

최대값 = -858993472.0000000

← 잘못된
실행 결과

함수의 기본_함수의 선언

- ❖ 함수의 선언이 있으면 함수가 정의된 위치에 관계없이 함수를 호출할 수 있다



함수의 기본_함수의 선언

❖ 함수의 선언을 함수의 **원형 (prototype)** 또는 **시그니처 (signature)**라고도 한다.

형식

리턴형 함수명(매개변수목록);

사용예

```
int add(int x, int y);  
double get_area(double radius);
```

함수의 정의

함수의 정의에서 헤더 부분을 복사한다.

```
double get_area(double radius)  
{  
    const double pi = 3.14;  
    return pi * radius * radius;  
}
```

함수의 선언

세미콜론을 붙여서 함수의 선언문을 만든다.

```
double get_area(double radius);
```

```
double get_area(double);
```

함수 선언에서 매개변수 이름은 생략할 수 있다.

함수의 기본_함수의 선언

❖ 함수의 선언에서 매개변수의 이름은 생략할 수 있다.

- 매개변수의 데이터형만 콤마(,)로 나열한다.

```
void draw_line(char, int);
```

- 함수 선언문의 끝에 세미콜론(;)을 써주어야 한다.

❖ 함수 선언문은 일반적으로 소스 파일의 시작 부분에 넣어준다.

함수의 기본_함수의 선언

❖ 함수의 선언을 사용하는 경우의 예(1/2)

```
01  #include <stdio.h>
```

```
04  void draw_line(char, int);
```

```
05  double get_area(double radius);
```

```
07  int main(void)
```

```
08  {
```

```
09      int r;
```

```
10
```

```
11      draw_line('-', 40);
```

```
12      for (r = 5; r <= 20; r+=5)
```

```
13          printf("r=%d, 원의 면적: %.2f\n", r, get_area(r));
```

```
14      draw_line('-', 40);
```

```
15  }
```

매개변수 이름은 생략할 수 있다.

함수의 선언을 소스 파일의 시작 부분에 써준다.

함수 선언이 있으므로 함수를 호출할 수 있다

함수의 기본_함수의 선언

❖ 함수의 선언을 사용하는 경우의 예(2/2)

```
17  double get_area(double radius)
18  {
19      const double pi = 3.14;
20      return pi * radius * radius;
21  }
22
23  void draw_line(char ch, int len)
24  {
25      int i;
26      for (i = 0; i < len; i++)
27          printf("%c", ch);
28      printf("\n");
29  }
```

실행 결과

```
-----
r=5, 원의 면적: 78.50
r=10, 원의 면적: 314.00
r=15, 원의 면적: 706.50
r=20, 원의 면적: 1256.00
-----
```

함수의 기본_정리

❖ 함수의 정의(definition)

- 함수의 리턴형, 함수의 이름, 함수의 매개변수를 써준 다음 {} 안에 실제로 함수가 처리할 내용을 기술

❖ 함수의 호출(call)

- 앞에서 선언되거나 정의된 함수를 이용
- 인자를 넘겨주고 리턴 값을 받아올 수 있다.

❖ 함수의 선언(declaration)

- 함수의 내용을 알려주지는 않지만 함수 호출에 필요한 리턴형, 함수의 이름, 매개변수 정보를 미리 알려준다.

❖ 함수의 선언만 생략 가능

함수의 선언과 정의, 호출

```
int GetFactorial ( int );
```

함수의 선언

```
int main(void)
{
    ...
```

```
    result = GetFactorial ( 10 );
```

함수의 호출

```
    return 0;
}
```

```
int GetFactorial ( int num )
{
    int i;
    int fact = 1;
    for(i = 1 ; i <= num ; i++)
        fact *= i;
    return fact;
}
```

함수의 정의

예제

- ❖ ex1. 두 수 중에서 큰 수를 찾아서 반환하는 함수
 - 함수 반환형 : int, 함수명 : get_max, 매개변수 : int x, int y
- ❖ ex2. 사용자로부터 정수값을 입력받아 반환하는 함수
 - 함수 반환형 : int, 함수명 : get_integer, 매개변수 없음
- ❖ ex3. 거듭제곱을 계산하여 반환하는 함수
 - 함수 반환형 : double, 함수명 : power, 매개변수 : int x, int y
- ❖ ex4. 원의 둘레와 넓이를 구하여 반환하는 함수
 - 원주율(3.141592)은 매크로 상수로 정의해서 사용
 - 반지름은 get_integer 함수를 통해 입력받을 것!!
 - 함수 반환형 : double, 함수명 : get_peri, get_area
 - 매개변수 : radius

예제

❖ ex5. 직각 삼각형 둘레와 넓이 구하여 출력하는 함수(리턴값 없음)

- 밑변과 높이는 ger_integer 함수를 통해 입력받을 것!!
- 제곱근 구하는 함수는 sqrt() 사용, #include <math.h> 포함

❖ ex6. 2차 방정식의 근을 계산하는 함수 quad_eqn 작성

- quad_eqn() 함수는 2차 방정식 계수 a, b, c를 나타내는 double형의 3개의 매개변수를 입력받는다. 판별식이 양수인 경우에만 근을 출력하고, 만약 판별식의 값이 음수이면 근이 없다는 메시지를 출력하는 프로그램을 작성하라.
- 반환형은 없음, 판별식(discriminant): $b^2 - 4ac$, 근의 공식 : $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

예제

- ❖ 다음 프로그램을 정수를 입력받는 `get_integer` (return값 있음), 짝수/홀수 구분하는 `odd_even` 함수(return값 없음)를 정의하여 다시 작성하시오.

```
#include <stdio.h>

int main(void)
{
    int num;

    printf("정수를 입력하세요 : ");
    scanf("%d", &num);

    if( num == 0 )
        printf("%d는 짝수도 홀수도 아닙니다\n", num);
    else if ( num % 2 == 0 )
        printf("%d는 짝수입니다.\n", num);
    else
        printf("%d는 홀수입니다.\n", num);

    return 0;
}
```

지역 변수와 전역 변수

❖ 지역 변수

- 함수 안에 선언된 변수
- 지역 변수가 선언된 함수 안에서만 사용될 수 있다.

❖ 전역 변수

- 함수 밖에 선언된 변수
- 여러 함수에서 사용될 수 있는 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 | 쓰레기값 | 0으로 초기화 |

지역 변수와 전역 변수_지역변수

❖지역 변수(1/6)

- 함수 안에서 만들어지고, 함수 안에서만 사용되는 변수
- 지역 변수는 함수가 리턴할 때 항상 자동으로 없어진다.

```
int main(void)
{
    int count = 0; ○———— 지역 변수 count를 선언한다.
    printf("main: count = %d\n", count);
    return 0;
} ○———— 함수를 빠져나갈 때 지역 변수인 count가 소멸된다.
```

- 변수는 함수의 시작 부분이나 블록의 시작 부분에서 선언할 수 있다.

지역 변수와 전역 변수_지역변수

❖ 지역 변수(2/6)

- 지역 변수의 사용 범위는 지역 변수가 선언된 위치에 따라 결정된다.
 - 지역 변수는 선언된 블록 안에서만 사용 가능하다.
 - 다른 함수에 선언된 지역 변수는 사용할 수 없다.

함수 안에 선언하는 경우

```
int get_gcd(int x, int y)
{
    int r;
    while (y != 0)
    {
        r = x % y;
        x = y;
        y = r;
    }
    printf("%d", r); //OK
    return x;
}
```

함수 전체에서 사용할 수 있다.

사용 가능

while 블록 안에 선언하는 경우

```
int get_gcd(int x, int y)
{
    while (y != 0)
    {
        int r;
        r = x % y;
        x = y;
        y = r;
    }
    printf("%d", r); // ERROR
    return x;
}
```

while 블록 안에서만 사용할 수 있다.

while 블록 밖에서는 사용할 수 없다.

지역 변수와 전역 변수_지역변수

❖ 다른 함수에 선언된 변수의 사용

```
01: /* 다른 함수에 선언된 변수 */
02: #include <stdio.h>
03:
04: void PrintCount(void);
05:
06: int main(void)
07: {
08:     int count = 0;
09:
10:     printf("main: count = %d\n", count);
11:
12:     return 0;
13: }
14:
15: void PrintCount(void)
16: {
17:     printf("PrintCount: count = %d\n", count);
18: }
```

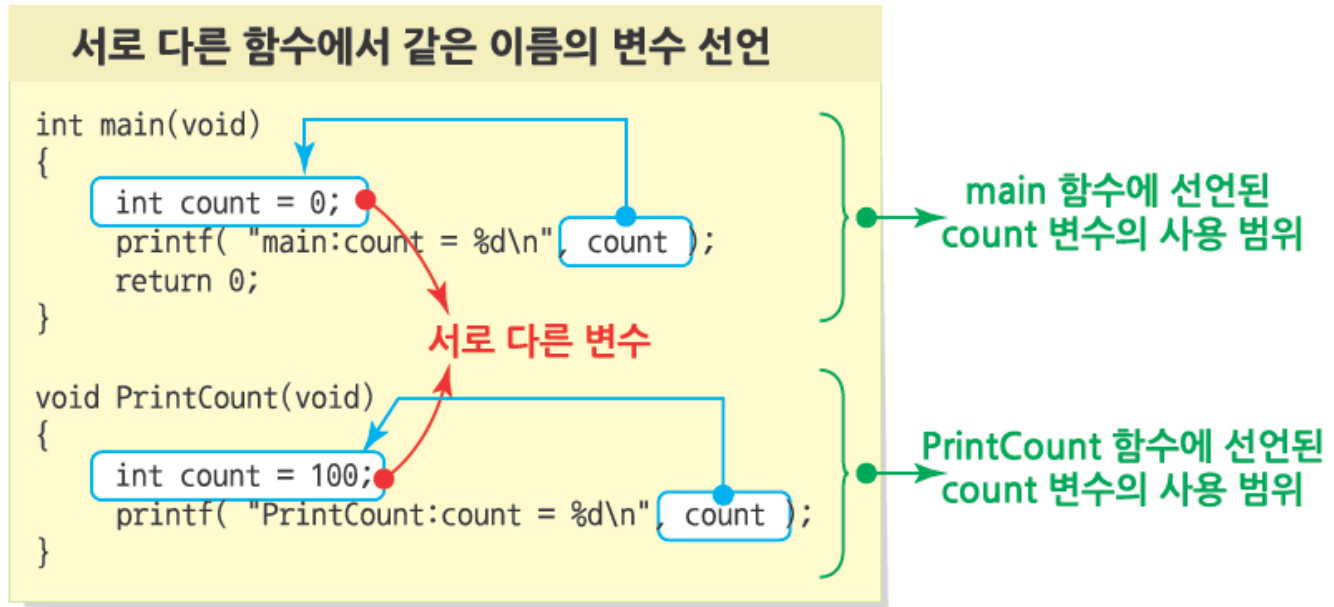
← **main 함수 안에서는 count 사용 가능**

← **PrintCount 함수 안에서 count 사용시 컴파일 에러**

지역 변수와 전역 변수_지역변수

❖ 지역 변수(3/6)

- 서로 다른 함수에서 같은 이름의 변수를 선언하면, 이름은 같지만 서로 다른 변수가 된다.



- 지역 변수는 따로 초기화하지 않으면 쓰레기 값을 갖는다.

지역 변수와 전역 변수_지역변수

❖ 서로 다른 함수에서 같은 이름의 변수를 선언하는 경우

```
01: /* 서로 다른 함수에서 같은 이름의 변수를 선언 */
```

```
02: #include <stdio.h>
```

```
03:
```

```
04: void PrintCount(void);
```

```
05:
```

```
06: int main(void)
```

```
07: {
```

```
08:     int count = 0; ← main 함수 안에 지역 변수 count 선언
```

```
09:     printf("main: count = %d\n", count); ← main 함수 안에 선언된 count 사용
```

```
10:
```

```
11:     PrintCount( );
```

```
12:
```

```
13:     return 0;
```

```
14: }
```

```
15:
```

```
16: void PrintCount(void)
```

```
17: {
```

```
18:     int count = 100; ← PrintCount 함수 안에 지역 변수 count 선언
```

```
19:     printf("PrintCount: count = %d\n", count); ← PrintCount 함수 안에 선언된
```

```
20: }
```

count 사용

실행 결과

```
main: count = 0
```

```
PrintCount: count = 100
```

지역 변수와 전역 변수_지역변수

❖ 지역 변수(4/6)

- 함수 안에 선언된 지역 변수는 함수가 호출되는 횟수만큼 생성되고 소멸된다.
- 반복문의 블록 안에 선언된 지역 변수는 반복문이 수행되는 횟수만큼 생성되고 소멸된다.

```
void test(void)
{
    int num = 100;
    printf("num = %d\n", num++);
}
```

함수가 호출될 때
매번 다시 생성되고
초기화된다.

함수가 리턴할 때
num이 소멸된다.

```
for (i = 0; i < 10; i++) {
    int num = 100;
    printf("num = %d\n", num--);
}
```

블록을 시작할 때
생성된다.

블록의 끝을 만나면
소멸된다.

지역 변수와 전역 변수_지역변수

❖ 같은 함수를 여러 번 호출하는 경우

```
01: /* 같은 함수를 여러 번 호출 */
```

```
02: #include <stdio.h>
```

```
03:
```

```
04: void TestLocal(void);
```

```
05:
```

```
06: int main(void)
```

```
07: {
```

```
08:     TestLocal( );
```

```
09:     TestLocal( );
```

```
10:
```

```
11:     return 0;
```

```
12: }
```

```
13:
```

```
14: void TestLocal(void)
```

```
15: {
```

```
16:     int num = 0; ← 지역 변수 num은 함수가 호출될 때마다 새로 생성
```

```
17:
```

```
18:     printf("num = %d\n", num++);
```

```
19: }
```

실행 결과

```
num = 0
```

```
num = 0
```

← 같은 함수를 여러 번 호출

지역 변수와 전역 변수_지역변수

❖ 지역 변수(5/6)

- 함수의 매개변수도 지역 변수이다.
- 매개변수를 변경하는 것은 아무 의미가 없다.
 - 매개변수의 값을 변경해도 함수가 리턴할 때 소멸되기 때문

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

int main(void)
{
    int x = 10;
    double_it(x);
    printf("x = %d\n", x);
}
```

매개변수를 변경하는 것은 아무 의미 없다.

함수가 리턴할 때 num이 소멸된다.

num은 x의 복사본으로 볼 수 있다.

double_it 함수가 호출된 다음에도 x의 값은 변경되지 않는다.

지역 변수와 전역 변수_지역변수

❖지역 변수(6/6)

- 함수 안에서 만들어진 값(지역변수)을 함수를 호출하는 쪽에 넘겨 주려면, 리턴값을 이용해야 한다.

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

리턴값을 전달한 다음
에 num이 소멸된다.

리턴값을 함수를 호출한
곳으로 먼저 전달한다.

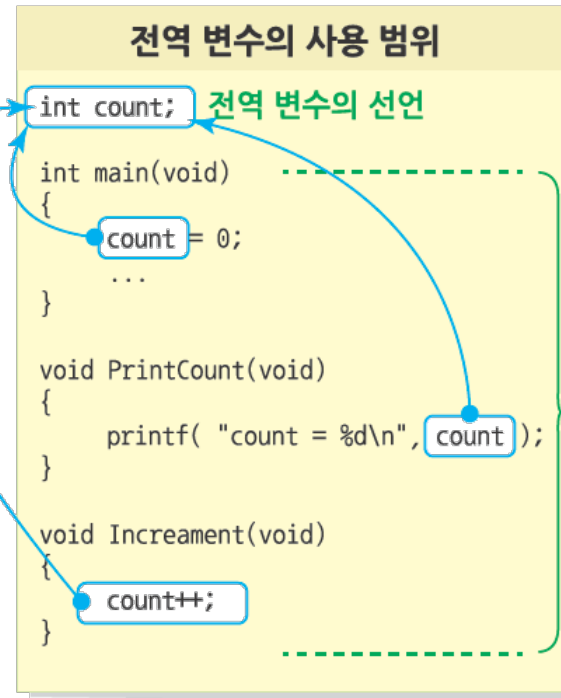
```
int main(void)
{
    int x = 10;
    x = double_this(x);
    printf("x = %d\n", x);
}
```

함수 안에서 생성된
값을 받아올 수 있다.

지역 변수와 전역 변수_전역변수

❖ 전역 변수(1/2)

- 프로그램 전체에서 사용될 수 있는 변수
- 프로그램이 시작될 때 한 번만 생성되고, 프로그램이 수행되는 동안 여러 함수에서 사용되다가, 프로그램이 종료될 때 비로소 해제된다.



전역 변수의
사용 범위

지역 변수와 전역 변수_전역변수

❖ 전역 변수의 선언 및 사용(1/2)

```
01: /* 전역변수의 선언 및 사용 */
02: #include <stdio.h>
03:
04: void PrintCount(void);
05: void Increment(void);
06: void Decrement(void);
07:
08: int count; ← 전역 변수 count의 선언
09:
10: int main(void)
11: {
12:     count = 0; ← 전역 변수 count의 사용
13:
14:     PrintCount( );
15:     Increment( );
16:     Increment( );
17:     PrintCount( );
18:
```

지역 변수와 전역 변수_전역변수

❖ 전역 변수의 선언 및 사용(2/2)

```
19:      Decrement( );
20:      PrintCount( );
21:
22:      return 0;
23: }
24:
25: void PrintCount(void)
26: {
27:     printf("count = %d\n", count); ← 전역 변수 count 의 사용
28: }
29:
30: void Increment(void)
31: {
32:     count++; ← 전역 변수 count 의 사용
33: }
34:
35: void Decrement(void)
36: {
37:     count--; ← 전역 변수 count 의 사용
38: }
```

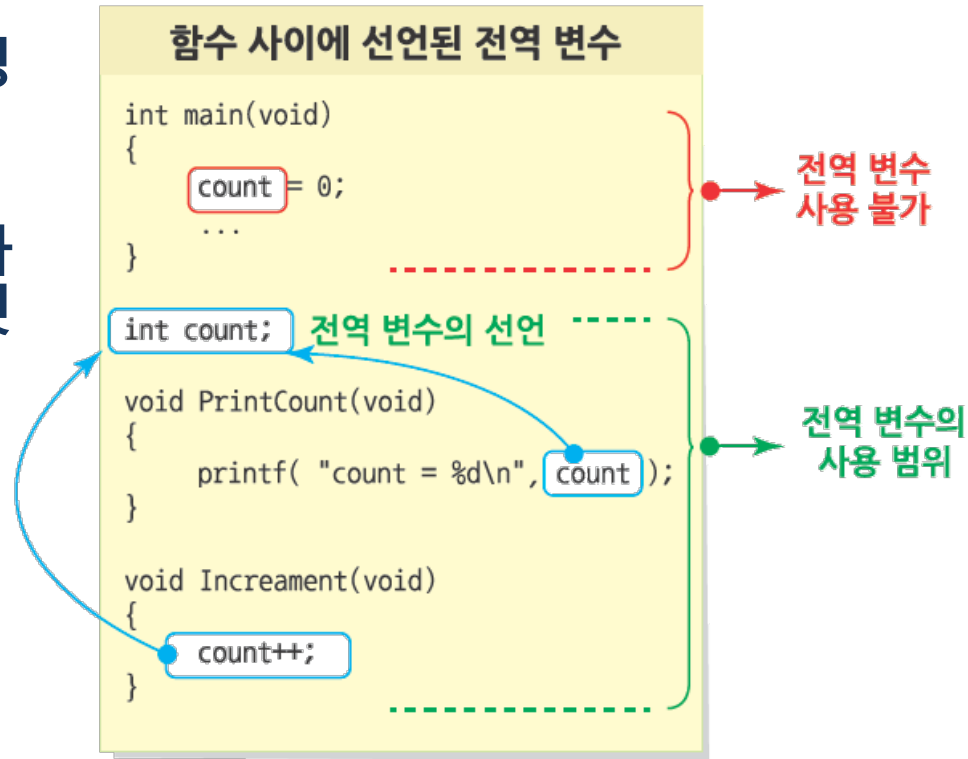
실행 결과

```
count = 0
count = 2
count = 1
```


지역 변수와 전역 변수_전역변수

❖ 전역 변수(2/2)

- 전역 변수의 선언문 다음에 정의된 함수에서만 전역 변수를 사용할 수 있다.
- 전역 변수의 선언문은 소스 파일의 시작 부분에 넣어주는 것이 좋다.
- 전역 변수는 따로 초기화하지 않으면 자동으로 0으로 초기화된다.



지역 변수와 전역 변수_전역변수

❖ 함수 사이에 선언된 전역 변수(1/2)

```
01: /* 함수 사이에 선언된 전역변수 */
02: #include <stdio.h>
03:
04: void PrintCount(void);
05: void Increment(void);
06: void Decrement(void);
07:
08: int main(void)
09: {
10:     count = 0;
11:
12:     Increment( );
13:     PrintCount( );
14:
15:     return 0;
16: }
17:
18: int count;
19:
```

선언 전에 전역 변수 **count**를 사용했으므로 컴파일 에러

전역 변수 **count**의 선언

10번째 줄에서 컴파일
에러가 발생하므로
실행할 수 없다!

지역 변수와 전역 변수_전역변수

❖ 함수 사이에 선언된 전역 변수(2/2)

```
20: void PrintCount(void)
21: {
22:     printf("count = %d\n", count); ← 전역 변수 count 의 사용
23: }
24:
25: void Increment(void)
26: {
27:     count++; ← 전역 변수 count 의 사용
28: }
29:
30: void Decrement(void)
31: {
32:     count--; ← 전역 변수 count 의 사용
33: }
```

지역 변수와 전역 변수_변수의 영역규칙

❖ 변수의 영역 규칙(1/2)

- 블록 범위가 달라질 때는 같은 이름의 변수를 여러 번 선언할 수 있다.

```
int main(void)
{
    int num = 10;
    if( 1 )
    {
        int num = 20;
        ...
    }
    while( 1 )
    {
        int num = 30;
    }
}
```

블록 안에서는 같은 이름의 변수를 재선언할 수 있다.

블록 안에서는 같은 이름의 변수를 재선언할 수 있다.

- 항상 가까운 블록 안에 선언된 변수 이름이 우선적으로 사용된다.

지역 변수와 전역 변수_변수의 영역규칙

❖ 변수의 영역 규칙(2/2)

- 같은 이름의 지역 변수가 있을 때는 전역 변수를 사용할 수 없다.
- 전역 변수와 지역 변수의 이름이 같아지는 것을 막기 위해서 전역 변수 이름에는 **g_**를 접두사로 붙인다.

```
int g_num;
int main(void)
{
    ...
}
```

전역 변수라는 것을 명확히 알 수 있다.

변수의 영역 규칙

```
int num = 10; // 프로그램 전체 범위

int main(void) // main 함수 범위
{
    int num = 20;

    while( 1 ) // while 블록 범위
    {
        int num = 30;
        printf( "num = %d\n", num ++ );
        if ( num > 25 )
            break;
    }

    printf( "num = %d\n", num );
    Test();
    return 0;
}

void Test(void)
{
    printf( "num = %d\n", num );
}
```

지역 변수와 전역 변수_변수의 영역규칙

❖ 변수의 영역 규칙(1/2)

```
01: /* 변수 영역 규칙 */
02: #include <stdio.h>
03:
04: void Test(void);
05: int num = 10; ← 전역 변수 num의 선언
06:
07: int main(void)
08: {
09:     int num = 20; ← 지역 변수 num의 선언
10:
11:     while( 1 )
12:     {
13:         int num = 30; ← 지역 변수 num의 선언
14:
15:         printf("num = %d\n", num++); ← while 안에 선언된 지역 변수 num 사용
16:
17:         if(num > 25)
18:             break;
19:     }
```

지역 변수와 전역 변수_변수의 영역규칙

❖ 변수의 영역 규칙(2/2)

```
20:    printf("num = %d\n", num); ← main 안에 선언된 지역 변수 num 사용
21:
22:    Test( );
23:
24:    return 0;
25: }
26:
27: void Test(void)
28: {
29:    printf("num = %d\n", num); ← 전역 변수 num 사용
30: }
```

실행 결과

```
num = 30
num = 20
num = 10
```

함수의 인자 전달 방법

- ❖ 값에 의한 전달 방법
- ❖ 포인터에 의한 전달 방법

함수의 인자 전달 방법_값에 의한 전달

❖ 값에 의한 전달 방법

- 함수를 호출할 때 넘겨주는 인자의 값을 함수 정의에 있는 매개변수로 복사해서 전달하는 방식
- **복사에 의한 전달**
- 함수 안에서 매개변수의 값을 변경해도 함수를 호출한 곳에 있는 인자의 값은 바뀌지 않는다.

값에 의한 전달

```
int GetFactorial ( int num ) ;

int main(void)
{
    ...
    result = GetFactorial ( 5 );
    ...

    return 0;
}

int GetFactorial ( int num )
{
    ...
    return fact;
}
```

int num = 5;

함수 호출시
내부적으로
수행되는 문장

함수의 인자 전달 방법_값에 의한 전달

❖ Swap 함수의 예(1/2)

```
01: /* swap함수의 예-값에 의한 전달 */
02: #include <stdio.h>
03:
04: void Swap(int x, int y); ← Swap 함수의 선언
05:
06: int main(void)
07: {
08:     int a = 10;
09:     int b = 20;
10:
11:     printf("Swap 전의 a = %d, b = %d\n", a, b);
12:
13:     Swap(a, b); ← Swap 함수의 호출
14:
15:     printf("Swap 후의 a = %d, b = %d\n", a, b);
16:
17:     return 0;
18: }
19:
```

함수의 인자 전달 방법_값에 의한 전달

❖ Swap 함수의 예(2/2)

```
20: void Swap(int x, int y)
21: {
22:     int temp;
23:
24:     temp = x;
25:     x = y;
26:     y = temp;
27: }
```

← Swap 함수의 정의

실행 결과

Swap 전의 a = 10, b = 20

Swap 후의 a = 10, b = 20

두 변수의 값이 맞
교환되지 않았다.

함수의 인자 전달 방법_값에 의한 전달

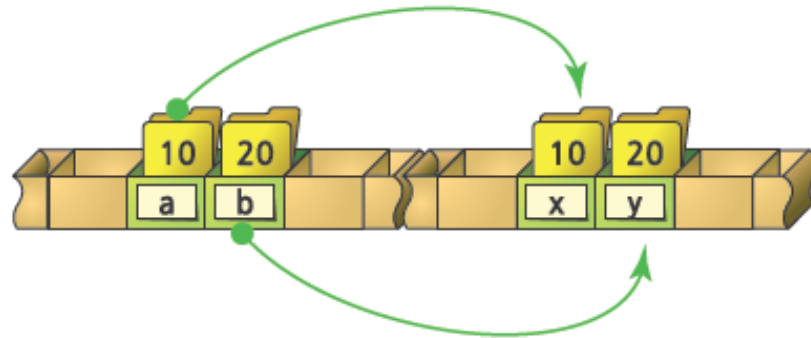
❖ Swap 함수의 수행 과정(1/2)

```
int main( void )
{
    ...
    Swap( a, b );
    ...
}

int x = a; int y = b;

void Swap( int x, int y )
{
    ...
}
```

① 함수 호출시 값에 의해 인자를 전달합니다.

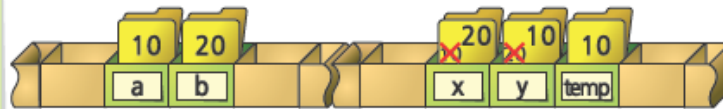


함수의 인자 전달 방법_값에 의한 전달

❖ Swap 함수의 수행 과정(2/2)

```
void Swap( int x , int y )  
{  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}
```

② Swap 함수의 코드를 수행합니다.



x와 y의 값을
맞교환합니다.

```
int main( void )  
{  
    ...  
    Swap( a , b );  
    ...  
}
```

③ Swap 함수를 리턴합니다.



a, b는 변경
되지 않습니다.

함수 리턴시
x, y, temp는 모두
해제됩니다.

함수의 인자 전달 방법_값에 의한 전달

❖ 매개변수로 함수의 처리 결과를 받아오는 경우

```
01: /* Ex06_15.c */
02: #include <stdio.h>
03:
04: void GetSmallerAndLarger(int a, int b, int smaller, int larger);
05:
06: int main(void)
07: {
08:     int smaller = 0, larger = 0; ← 결과를 받아올 변수의 선언
09:
10:     GetSmallerAndLarger(10, 20, smaller, larger); ← 함수 호출
11:
12:     printf("smaller = %d, larger = %d\n", smaller, larger);
13:
14:     return 0;
15: }
16:
17: void GetSmallerAndLarger(int a, int b, int smaller, int larger)
18: {
19:     smaller = a < b ? a : b;
20:     larger = a > b ? a : b;
21: }
```

작은 값과 큰 값이
구해지지 않았다!

실행 결과

smaller = 0, larger = 0

함수의 인자 전달 방법_포인터에 의한 전달

❖포인터에 의한 전달 방법

- 변수의 값을 전달하는 대신 변수의 주소를 전달하는 방식

학습 정리

❖ 함수의 기본

- 함수의 정의 : 함수의 리턴형, 함수의 이름, 함수의 매개변수를 써준 다음 {} 안에 실제로 함수가 처리할 내용을 기술한다.
- 함수의 호출 : 함수의 이름 다음에 ()안에 인자를 써준다.
- 함수의 선언 : 함수 호출에 필요한 리턴형, 함수의 이름, 매개변수 정보를 미리 알려주는 문장이다.

함수의 선언과 정의, 호출

```
int GetFactorial ( int );
```

함수의 선언

```
int main(void)  
{  
    ...
```

```
    result = GetFactorial ( 10 );
```

함수의 호출

```
    return 0;  
}
```

```
int GetFactorial ( int num )  
{  
    int i;  
    int fact = 1;  
    for(i = 1 ; i <= num ; i++)  
        fact *= i;  
    return fact;  
}
```

함수의 정의

학습 정리

❖ 지역 변수와 전역 변수

- 지역 변수 : 함수 안에서 선언되고, 함수 안에서만 사용된다.
 - 따로 초기화하지 않으면 쓰레기 값을 갖는다.
- 전역 변수 : 함수 밖에서 선언되고, 프로그램 전체에서 사용된다.
 - 따로 초기화하지 않아도 0으로 초기화된다.
- 영역 규칙 : 가장 가까운 블록에 선언된 변수가 항상 우선적으로 사용된다.

❖ 함수의 인자 전달 방법

- 값에 의한 전달 : 인자의 값을 매개변수에 복사해서 전달한다.
 - 함수 안에서 매개변수의 값을 변경해도 함수를 호출한 곳에 있는 인자의 값은 바뀌지 않는다.
- 포인터에 의한 전달 : 함수의 인자로 변수의 주소를 전달한다.