프로그래밍 기초

9장. 함수 기초 Part 1

학습목표

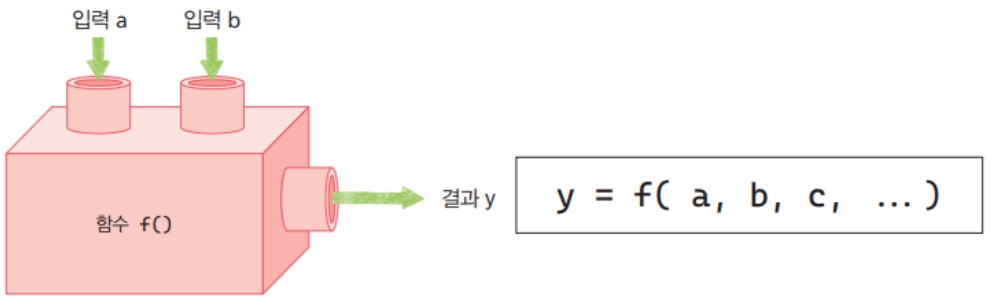
- **함수에 대해 이해하고 설명할 수 있다.**
  - 함수의 개념, 함수의 입력과 반환
  - 라이브러리와 사용자 정의 함수

- **함수를 사용하기 위한 함수 정의, 함수 호출, 함수원형을 이해하고 설명할 수 있다.**
  - 함수 정의에서의 함수머리와 함수 몸체
  - 함수 호출 방법 및 실행 순서
  - 함수원형 구문과 필요성
  - 함수로 배열 전달(1차원 배열, 2차원 배열)

함수의 이해

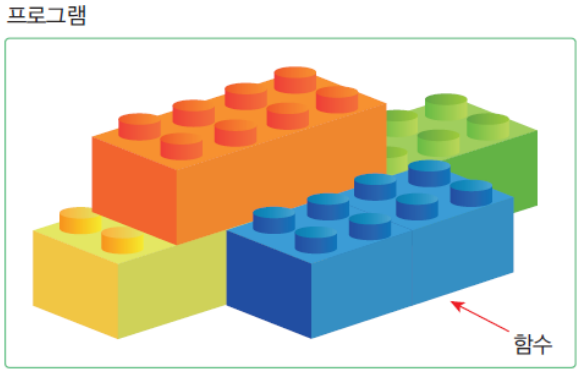
- **함수 개념**
  - 함수(function)
    - ➢ 특정한 작업을 수행하도록 설계된 독립된 프로그램 단위
      - – 필요한 입력을 받아 원하는 어떤 기능을 수행 후 결과를 반환(return)하는 프로 그램 단위
  - 함수 구분
    - ➢ **라이브러리 함수(library function)**
      - – printf(), scanf()와 같이 이미 개발 환경에 포함되어 있는 함수
    - ➢ **사용자 정의 함수(user defined function)**
      - – 필요에 의해서 개발자가 직접 개발하는 함수

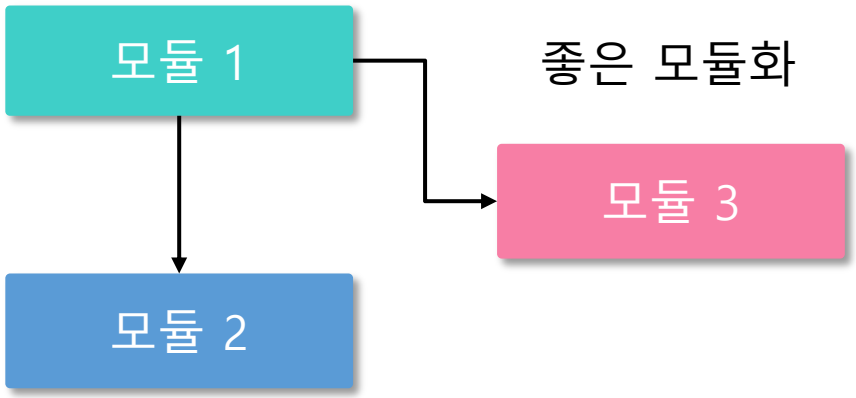
- **C 프로그램**
  - 여러 함수로 구성되는 프로그램
    - ➢ 최소한 main() 함수와 필요한 다른 함수로 구성
  - main() 함수
    - ➢ 프로그램의 실행이 시작되는 특수한 함수



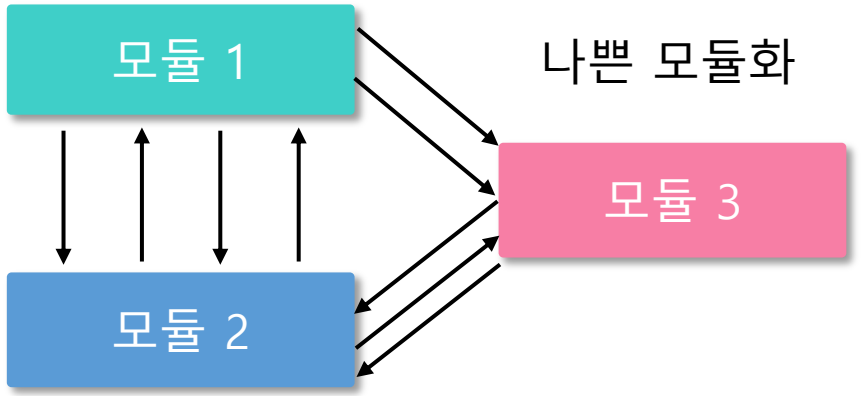
함수의 장점

- **함수 사용의 장점**
  - 코드가 중복되는 것을 막을 수 있음
  - 한번 작성된 함수는 여러 번 재사용할 수 있음
  - 함수를 사용하면 전체 프로그램을 모듈로 나누어서 구현
    - ➢ 프로그램이 체계적이고 유지 보수가 쉬어짐
    - ➢ 복잡한 문제를 단순한 부분으로 분해





좋은 모듈화



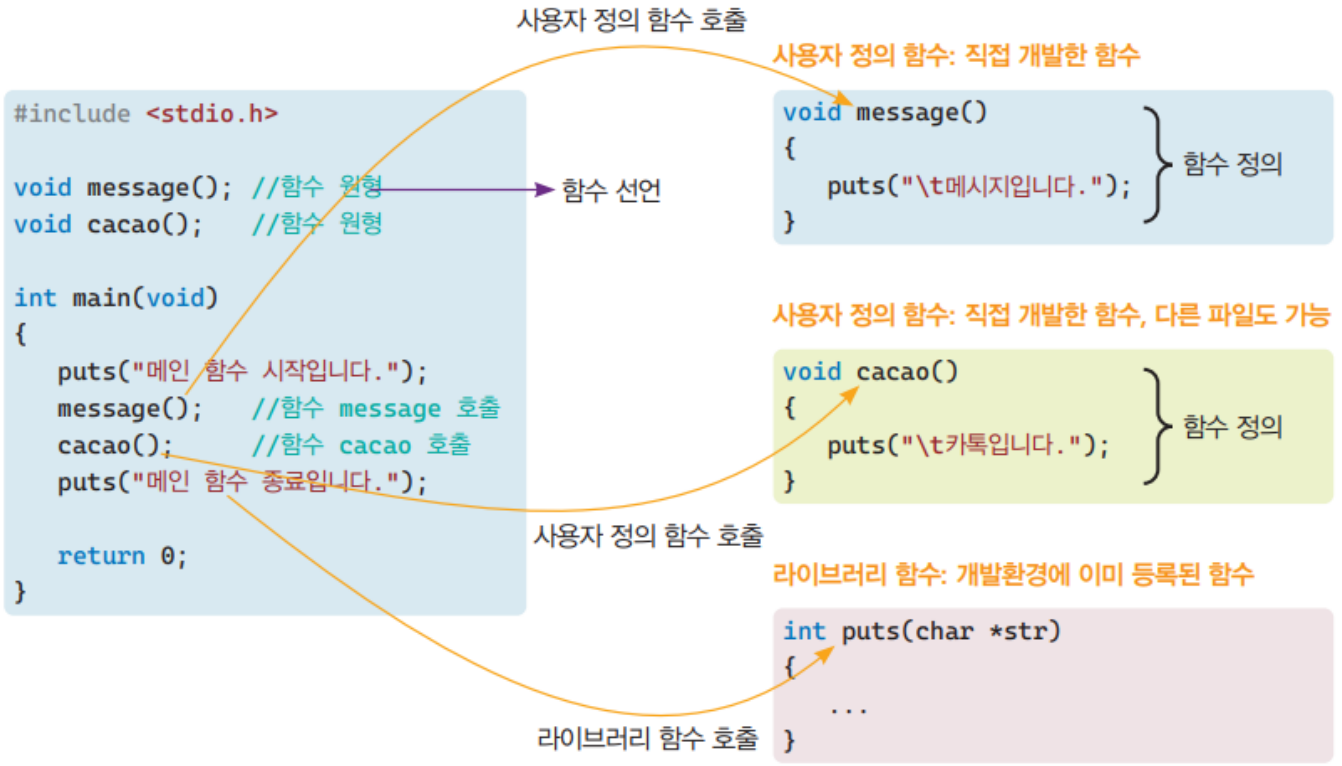
나쁜 모듈화

- 모듈2를 교체(수정)할 경우, 하나의 연결만 고려하면 됨
- 모듈 내부는 한 가지 작업만 수행
- 모듈과 모듈 사이는 최소의 연관성을 가짐

함수 정의와 호출

- **C 프로그램 main() 함수**
  - 첫 줄에서 시작하여 마지막 줄을 실행한 후 종료
  - 사용자가 직접 개발한 함수를 사용
    - 함수 선언(function declaration)
    - 함수 정의(function definition)
    - 함수 호출(function call)

- **하나의 응용 프로그램**
  - 하나의 main() 함수와 여러 개의 다른 함수로 구성
    - 필요에 따라 여러 소스 파일로 나누어 프로그래밍 가능



2개의 함수를 2개의 파일로 구현하고 있는 프로젝트 (실습)

- **2개의 함수 구현 (서로 다른 소스 파일에 구현)**
  - message() 함수: main()이 있는 01basefunc.c에 구현
  - cacao() 함수: 01catalk.c에서 구현

<01basefunc.c>

```c
#include <stdio.h>

extern void cacao();
void message();

int main(void)
{
    puts("메인 함수 시작입니다.");
    message(); // 함수 message 호출
    cacao(); // 함수 cacao 호출
    puts("메인 함수 종료입니다.");

    return 0;
}

// 함수 message 구현
void message()
{
    puts("\t메시지입니다.");
}
```

extern 키워드
- 해당 함수나 변수가 다른 파일에 정의되어 있음을 알림

함수 원형
- 해당 소스 파일의 상단에 선언

<01catalk.c>

```c
#include <stdio.h>

// 함수 main()과 다른 파일에 함수 구현 가능
void cacao()
{
    puts("\t카톡입니다.");
}
```

VS Code에서 Run C/C++ File 메뉴 선택
- Undefined symbols 에러 발생

```
Undefined symbols for architecture arm64:
  "_cacao", referenced from:
      _main in cclLGoNx.o
ld: symbol(s) not found for architecture arm64
collect2: error: ld returned 1 exit status
```

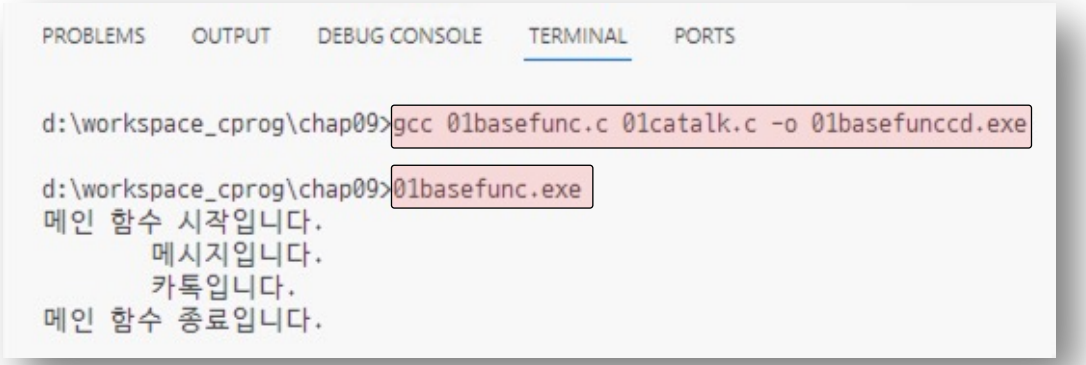
VS Code에서 여러 소스 파일 컴파일 하기

- gcc 컴파일 명령: Terminal에 직접 입력
  - -o 옵션: 컴파일 후 실행 파일 생성

- Windows 환경에서 컴파일 및 실행

```
gcc fil1.c file2.c –o 실행파일이름.exe Enter↵
```

```
실행파일이름.exe Enter↵
```

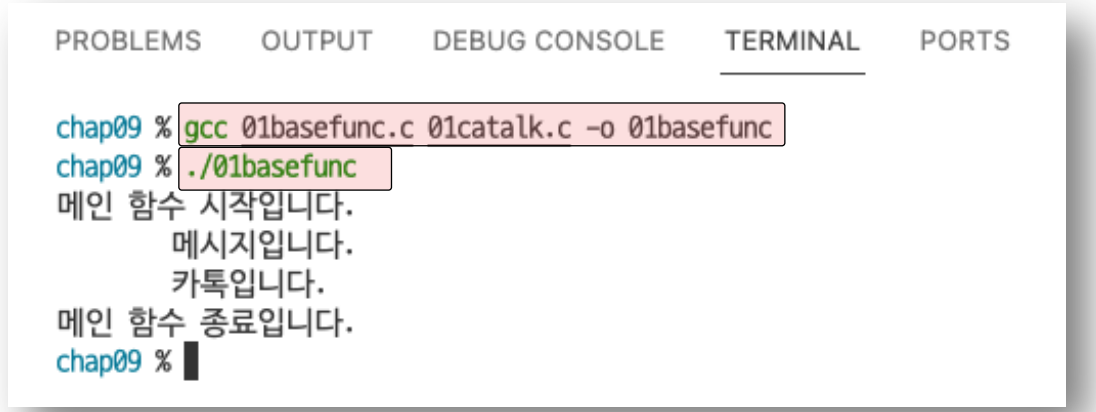


- Mac 환경에서 컴파일 및 실행
  - exe 확장자 필요 없음

```
gcc fil1.c file2.c –o 실행파일이름 Enter↵
```

```
./실행파일이름 Enter↵
```

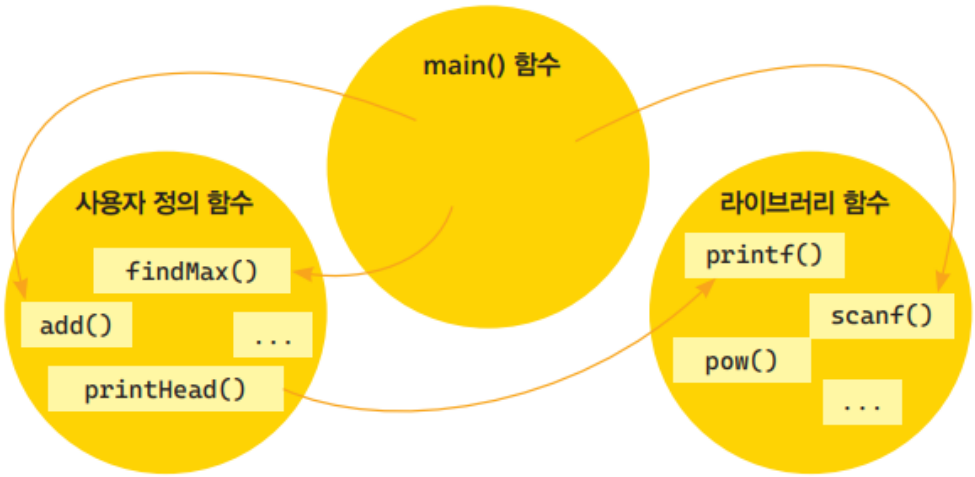
➢ ./: 현재 폴더를 의미(반드시 필요함)



절차적 프로그래밍

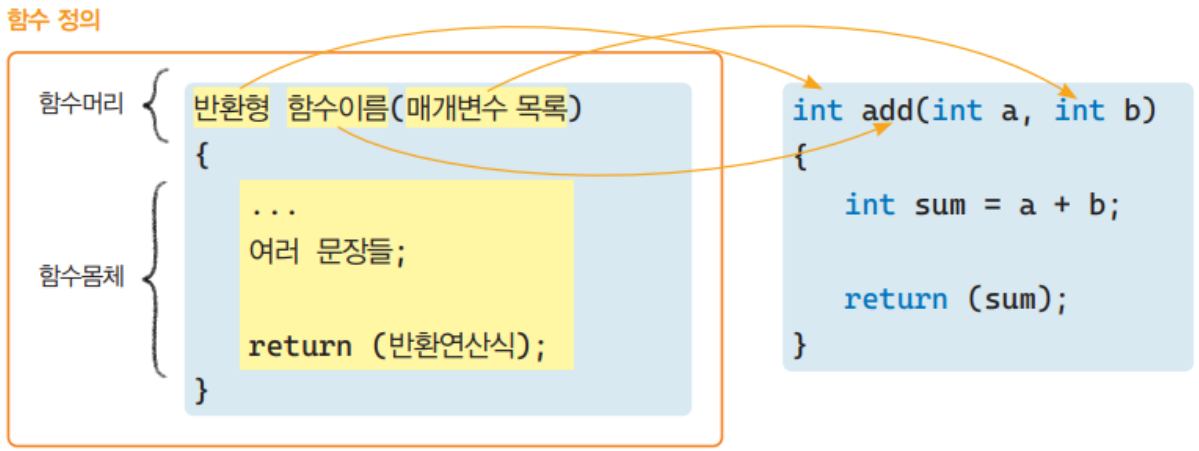
- **주어진 문제를 여러 개의 작은 문제로 나누어 해결 하듯**
  - 하나의 프로그램은 여러 함수로 나누어 프로그래밍
  - 문제 분해(problem decomposition)
    - 하나의 프로그램을 작은 단위의 여러 함수로 나누는 것
    - 문제 해결의 한 방법

- **절차적 프로그래밍(procedural programming) 방식**
  - 함수 중심의 프로그래밍 방식
    - 모듈화 프로그램 (modular program) 또는 구조화된 프로그램(structured program)
    - 적절한 함수로 잘 구성된 프로그램
      - 한번 정의된 함수는 여러 번 호출이 가능
      - 소스 중복을 최소화
    - 잘 구현된 함수
      - 여러 프로그램에서 손쉽게 이용이 가능: 재사용
      - 유지 관리도 편리



함수 정의 구문

- **함수 머리(function header)**
  - **반환형**과 **함수 이름**, **매개변수 목록**으로 구성
  - 반환형: 함수 결과값의 자료형
  - 함수 이름: 식별자의 생성 규칙
  - 매개변수 목록: '자료형 변수이름'의 쌍
    - ➢ 필요한 수만큼 콤마로 구분하여 기술
  - 예: `int add(int a, int b)`



- **함수 몸체(function body)**
  - 중괄호로 시작하여 중괄호로 종료: {. . .}
  - 함수가 수행해야 할 문장들로 구성
  - 마지막은 결과값을 반환하는 return 문장으로 종료
  - 결과값이 없다면 생략 가능

반환형(return type)

- **함수에서 반환 값을 전달하는 목적 및 함수의 종료를 알리는 문장**
  - 함수가 반환 값이 없다면 반환형으로 void를 기술

반환 값이 정수형(int)

```
int findMin(int x, int y)
{
    int min = x < y ? x : y;

    return min;
}
```

반환 값이 없는 경우

```
void printMin(int a, int b)
{
    int min = a < b ? a : b;
    printf("최소: %d\n", min);
    return;  // 생략
}
```

return 문장

```
return (반환연산식);
```

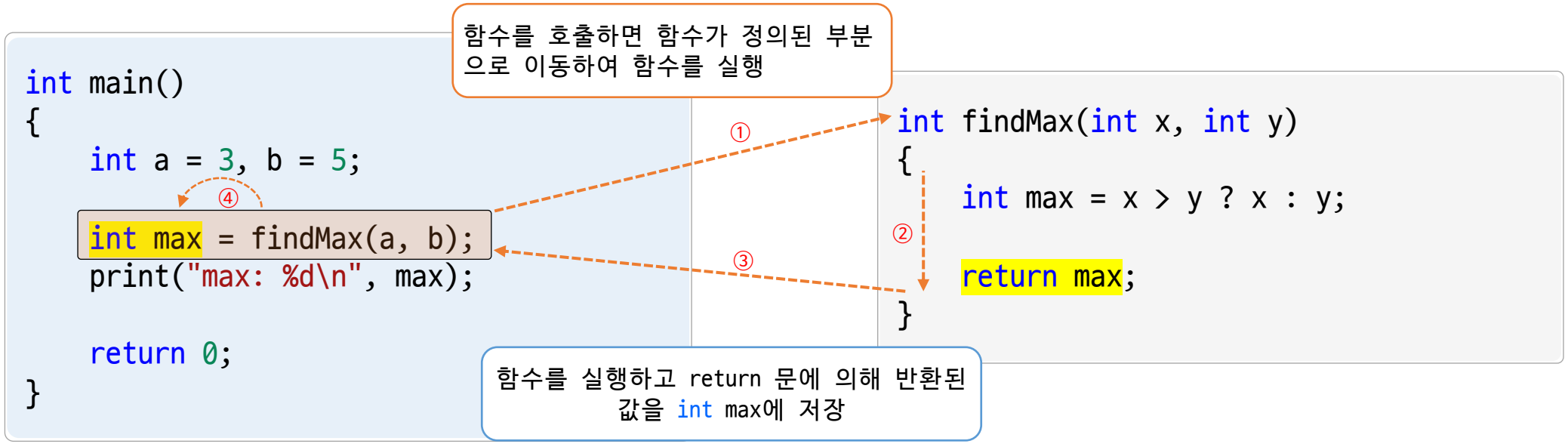
```
return 0;
return (a + b);
return 2 * PI * r;
return ;
```

함수 실행

정의된 함수 실행

- 프로그램 실행 중에 함수 호출(function call)이 필요
- 함수를 호출하려면 함수 이름과 함께 괄호 안에 적절한 매개변수가 필요
  - 즉 findMax(a, b), add(a, b), printMin(3, 5)와 같이 호출

- 함수 findMax(a, b)와 같이 반환 값이 있는 함수
  - 반환된 값을 저장하기 위해 변수 = 함수(a, b)
  - 문장 int max = findMax(3, 5);은 함수(findMax())의 반환 값을 변수 max에 저장



함수 원형

- **함수 원형(function prototype)**
  - 함수를 선언하는 문장
  - 정의된 함수를 호출하기 이전에 필요

- **함수 원형 구문**
  - 함수 머리에 세미콜론을 넣은 문장
    - `int add(int a, int b);`
    - `int add(int, int);` 도 가능

함수원형

반환형 함수이름(매개변수 목록);

함수원형은 문장이므로 마지막에 세미콜론 ;이 반드시 필요하다.

```
int add(int a, int b);
int add(int, int);
int findMin(int x, int y);
int findMin(int, int);
```

```
#include <stdio.h>

int add(int a, int b);   // 함수 원형
//int add(int, int);   // 매개변수 타입만 작성 가능

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

    int sum = add(a, b);   // 함수 호출
    printf("합: %d\n", sum);

    return 0;
}

int add(int a, int b)   // 함수 정의(구현)
{
    int sum = a + b;
    return sum;
}
```

함수 원형 선언 및 함수 선언 방법 2가지

```c
#include <stdio.h>
void myprintA(int a, int b);
void myprintB();
void myprintC();

int main()
{
    myprintA();
    myprintC();

    return 0;
}

void myprintA(int a, int b)
{
    printf("a: %d, b: %d\n", a, b);
}

void myprintB()
{
    printf("C Language! \n");
}

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

함수 원형 선언
- 함수 정의가 main() 함수 아래에 구현된 경우에는, 함수 원형 선언이 필요

함수 정의

```c
#include <stdio.h>

void myprintA(int a, int b)
{
    printf("a: %d, b: %d\n", a, b);
}

void myprintB()
{
    printf("C Language! \n");
}

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

int main()
{
    myprintA();
    myprintC();

    return 0;
}
```

함수 정의
- 함수 정의가 main() 함수 위쪽에 구현
- 함수 원형 선언이 필요 없음

함수의 정의와 호출 (실습)

<02funcadd.c>

```c
#include <stdio.h>

int add(int a, int b); // 함수 원형 선언
int findMin(int a, int b);

int main(void)
{
    int a = 3, b = 5;
    // int add(int a, int b); 도 가능

    // 위 함수원형이 없으면 함수호출에서 경고 발생
    int sum = add(a, b);
    printf("합: %d\n", sum);


    printf("작은 값: %d\n", findMin(a, b));
    return 0;
}
```

```c
// 함수 add의 함수구현 또는 함수정의 부분
int add(int a, int b)
{
    int sum = a + b;

    return sum;
}

int findMin(int x, int y)
{
    int min = x < y ? x : y;

    return min;
}
```

```
합: 8
작은 값: 3
```

LAB 함수 intpow()

<lab1intpower.c>

- **표준입력으로 받은 두 양의 정수로 거듭제곱을 구하는 함수**
  - intpow(int m, int n)
    - ➢ m의 n 제곱승을 구하는 함수

```
#include <stdio.h>

int intpow(int m, int n);
int main(void)
{
    int m, n;

    printf("정수 m을 n번 제곱합니다.\n");
    printf("정수 m n을 계속 입력 >>> ");
    scanf("%d %d", &m, &n);

    // 함수호출
    printf("\n%d의 %d 제곱은 %d입니다.\n", m, n, intpow(m, n));
    return 0;
}

int intpow(int m, int n)
{
    int mult = 1, i = 1;

    for (i = 1; i <= n; i++)
    {
        mult *= m;
    }
    return mult;
}
```

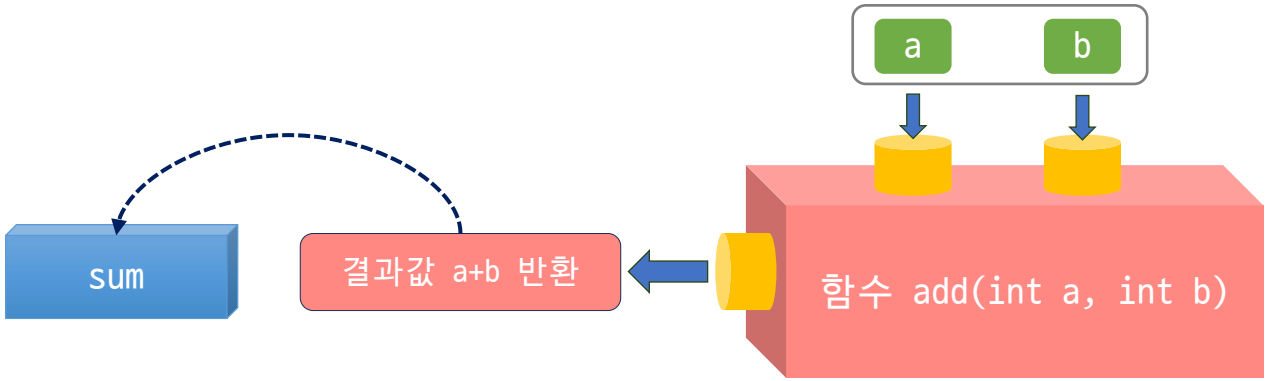
함수의 반환값 전달

실행 결과

```
수 m을 n번 제곱합니다.
정수 m n을 계속 입력 >>> 3 4

3의 4 제곱은 81입니다.
```

매개변수 정의

- **함수 매개변수(parameter)**
  - 함수를 호출하는 부분에서 함수 몸체로 값을 전달할 목적으로 이용
  - 자료형과 변수명의 목록으로 표시
  - 필요 없으면 키워드 void를 기술
- **반환값**
  - 함수를 호출한 영역으로 보내는 결과값을 전달



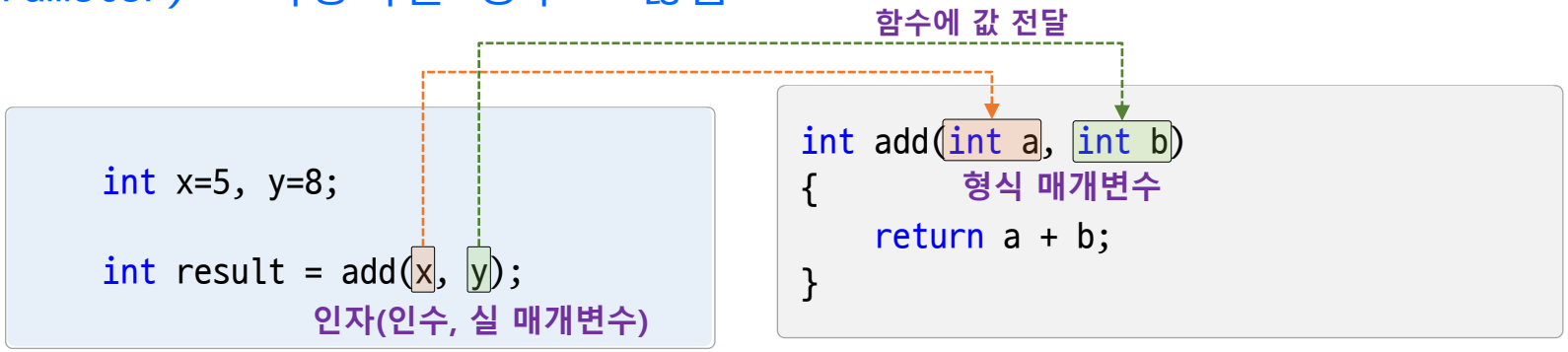

```
int sum = add(a, b);
```

형식매개변수와 실매개변수

- **형식 매개변수(formal parameter)**
  - 함수 정의 시 함수 헤더에 선언되는 변수
    - ➢ 함수 내부에서 입력 값을 받을 용도로 사용
  - 함수 내부에서만 사용될 수 있는 변수

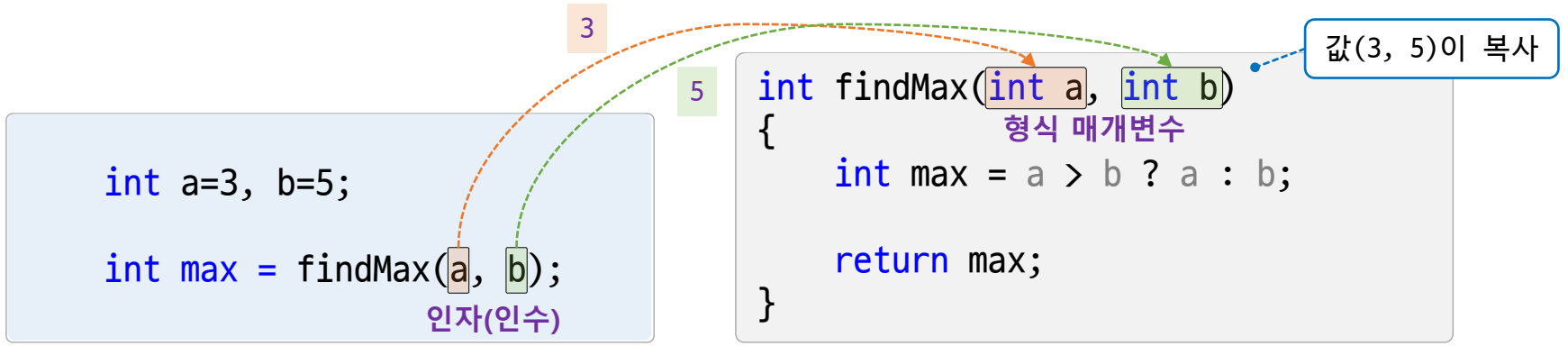
- **실 매개변수(real parameter) 또는 인자(argument)**
  - 함수 호출 시 전달되는 실제 값, 변수, 상수 혹은 표현식
  - 함수 외부에서 함수에 값을 전달

- **일반적으로 매개변수, 인자(인수)**
  - 구분하지 않고 매개변수(parameter)로 사용하는 경우도 많음



값에 의한 호출(call by value)

- **함수가 호출되는 경우 인자의 값이 형식 매개변수에 복사된 후 함수가 실행**
  - 함수 호출 findMax(a, b)에서 이용되는 인자 a, b는
    - ➢ 함수 정의 int findMax(int a, int b)에서 매개변수 a, b와는 전혀 다른 변수
  - 함수 내부에서 매개변수의 값을 수정하더라도
    - ➢ 함수를 호출한 곳(main() 함수 등)에서 실제 변수의 값은 변화되지 않음: Call by value의 특징




```
int max = findMax();          // 오류 발생
int max = findMax(2);         // 오류 발생
int max = findMax(2.4);       // 오류 발생
int max = findMax(2.4, 6.8); // 오류 발생
```

함수 호출 시 매개변수의 수와 자료형이 다르면 문법 오류 발생

함수의 정의와 호출: 2개의 소스 파일 (실습)

<03functioncall.c>

```c
#include <stdio.h>

extern int add(int a, int b);
extern int findMax(int, int);
extern int findMin(int, int);
void printMin(int, int);

int main(void)
{
    int a = 10, b = 15;

    int max = findMax(a, b);
    printf("최대: %d\n", max);
    printf("합: %d\n", add(a, b));

    // 반환 값이 없는 함수호출은 일반문장처럼 사용
    printMin(a, b);

    return 0;
}

void printMin(int a, int b)
{
    int min = a < b ? a : b;
    printf("최소: %d\n", min);
}
```

<03others.c>

```c
//함수 add, findMax, findMin, printMin 구현
int add(int a, int b)
{
    int sum = a + b;

    return sum;
}

int findMax(int a, int b)
{
    int max = a > b ? a : b;

    return max;
}

int findMin(int x, int y)
{
    int min = x < y ? x : y;

    return min;
}
```

```
gcc 03functioncall.c 03others.c -o 03functioncall.exe
```

```
최대: 15
합: 25
최소: 10
```

배열 주소 확인 (실습)

- **배열의 이름(point)**
  - point: 배열의 시작 주소를 나타냄
  - &point[i]: 배열 인덱스[i]의 시작 주소
    - ➢ &: 주소 연산자

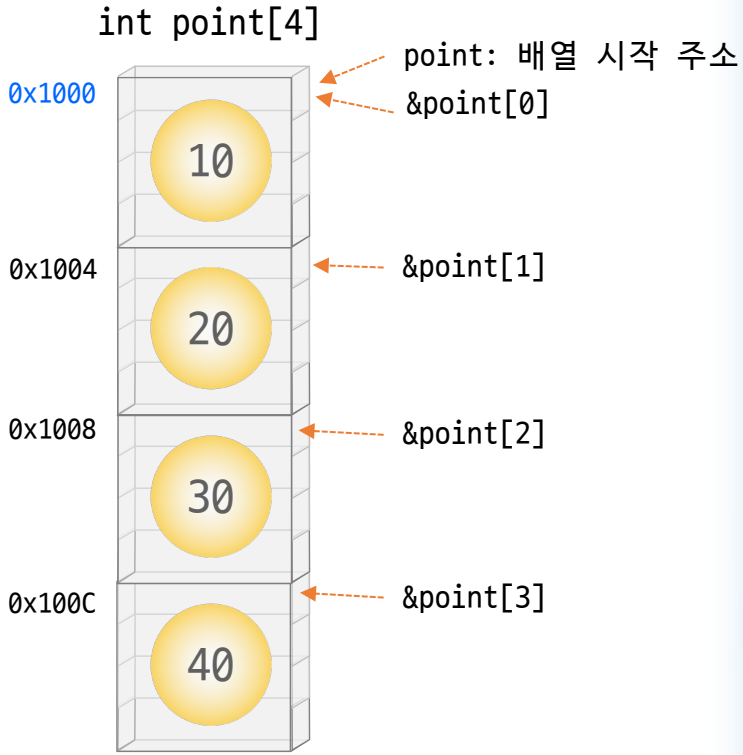
<04arrayaddr.c>

```c
#include <stdio.h>

int main()
{
    int point[4] = {10, 20, 30, 40};

    printf("point : %#x\n", point);
    printf("&point[0]: %#x\n", &point[0]);

    return 0;
}
```

배열 point의 시작 주소

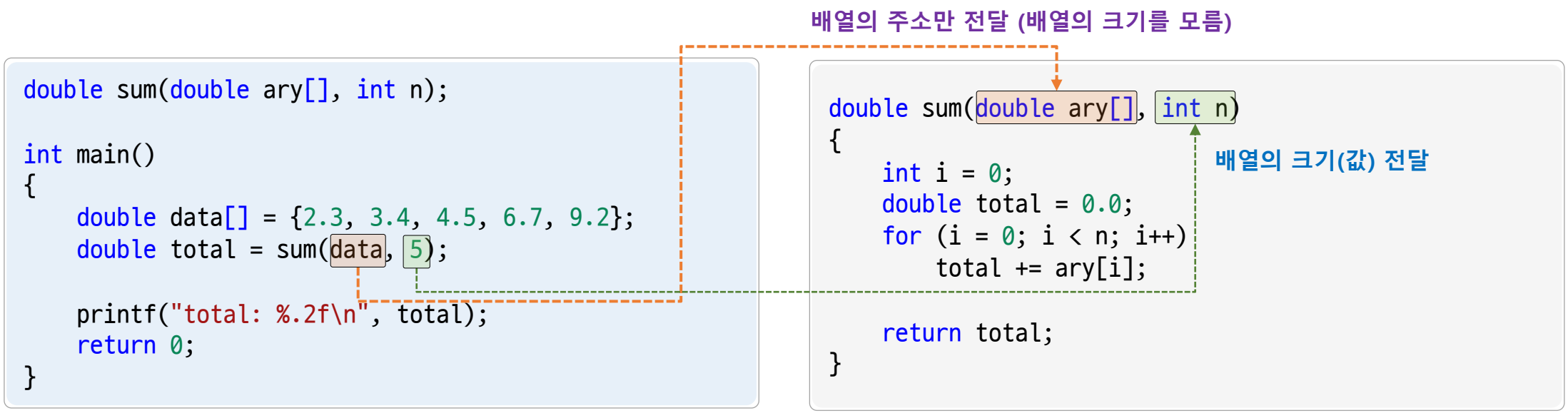
```
point    : 0x6d99ea90
&point[0]: 0x6d99ea90
```

주소가 동일
(point, &point[0])



매개변수로 배열 사용

- **함수의 매개변수로 배열을 전달**
  - 한 번에 여러 개의 변수를 전달하는 효과
- **함수 sum()**
  - 형식 매개변수는 배열(double ary[])과 배열 크기(int n)
    - 첫 번째 형식 매개변수에서 배열 자체에 배열 크기를 기술하는 것은 아무 의미가 없음
    - double ary[5]보다는 double ary[]라고 기술하는 것을 권장
  - 실제로 함수 내부에서 실인자로 전달되는 배열크기를 알 수 없음(주소만 전달)
    - 배열 크기를 두 번째 인자로 사용



배열을 매개변수로 사용하는 방법

- **함수 선언에 배열을 매개변수로 사용:**
  - 배열 크기를 제외한 `int ary[]`로 기술
- **함수 호출: 배열 이름 `point` 또는 `&point[0]`를 사용**
  - 배열의 시작 주소만 전달: 배열의 크기를 알 수 없기 때문에
  - 배열의 크기도 매개변수로 함께 전달

```
printf("함수 sumary() 호출로 구한 합은 %d\n", summary(point, aryLength));
printf("함수 sumary() 호출로 구한 합은 %d\n", summary(&point[0], aryLength));
```

배열의 시작 주소 전달

`int ary[]`로 선언
- 배열의 시작 주소를 전달 받음

```
int summary(int ary[], int SIZE)
{
    for (int i = 0; i < SIZE; i++)
    {
        sum += ary[i];
    }
    return sum;
}
```

배열의 크기는 의미 없음, 배열 크기를 인자로 사용

```
int summary(int ary[10], int SIZE)
{
        . . .
}
```

함수에서 일차원 배열 인자를 사용 (실습)

<04aryparam.c>

```c
#include <stdio.h>

int summary(int ary[], int SIZE);

int main(void)
{
    int point[] = {95, 88, 76, 54, 85, 33, 65, 78, 99, 82};

    int arySize = sizeof(point);

    printf("메인에서 배열 전체크기: %d\n", arySize);
    int aryLength = arySize / sizeof(int);

    int sum = 0;
    for (int i = 0; i < aryLength; i++)
        sum += point[i];

    printf("메인에서 구한 합은 %d\n", sum);
    printf("함수 summary() 호출로 구한 합은 %d\n",
                                summary(point, aryLength));

    printf("함수 summary() 호출로 구한 합은 %d\n",
                                summary(&point[0], aryLength));

    return 0;
}
```

4 bytes x 10개 = 40 bytes
(실제 배열의 크기 계산)

배열 point의 원소 개수

```c
int summary(int ary[], int SIZE)
{
    int sum = 0;
    printf("함수에서 배열 전체크기: %zd\n", sizeof(ary));
    for (int i = 0; i < SIZE; i++)
    {
        sum += ary[i];
    }

    return sum;
}
```

주의: 실제 배열의 크기와 다름
(64bit 주소 사용: 8 bytes 리턴)

배열 point의 주소만 전달됨
- 배열의 크기를 알 수 없기 때문에 aryLength를 별도로 전달함

```
메인에서 배열 전체크기: 40
메인에서 구한 합은 755
함수에서 배열 전체크기: 8
함수 summary() 호출로 구한 합은 755
함수에서 배열 전체크기: 8
함수 summary() 호출로 구한 합은 755
```

이차원 배열을 함수 매개변수로 이용하는 방법

- **함수 원형과 함수 정의의 헤더**
  - 함수 선언 시 첫 번째 대괄호인 행(row)의 크기를 제외한 열(column)의 크기는 반드시 기술
    - 행(row)과 열(column)의 크기를 별도의 파라미터로 전달

```
double x[][2] = { { 11, 12 },
                  { 21, 22 },
                  { 31, 32 } };


int rowsize = sizeof(x) / sizeof(x[0]);
int colsize = sizeof(x[0]) / sizeof(x[0][0]);


printarray(x, rowsize, colsize);

printf(" %.3lf \n", sum(x, rowsize, colsize));
```

함수로 2차원 배열 전달
- 배열의 이름만 전달 (주소 전달)
- 배열의 크기를 별도로 전달해야 됨

2차원 배열의 column(열)의 크기 입력

```
double printarray(double data[][2], int rowsize, int colsize)
{
        . . .

    return total;
}
```

2차원 배열의 column(열)의 크기 입력

```
double sum(double data[][2], int rowsize, int colsize)
{
        . . .

    return total;
}
```

2차원 배열을 인자로 하는 함수의 정의와 호출

<05twodaryfunc.c>

```c
#include <stdio.h>

//2차원 배열값을 모두 더하는 함수원형
double sum(double data[][2], int, int);

//2차원 배열값을 모두 출력하는 함수원형
void printarray(double data[][2], int, int);

int main(void)
{
    //3 x 2 행렬을 위한 이차원 배열 선언 및 초기화
    double x[][2] = { { 11, 12 }, { 21, 22 }, { 31, 32 } };

    int rowsize = sizeof(x) / sizeof(x[0]);
    int colsize = sizeof(x[0]) / sizeof(x[0][0]);

    printf("2차원 배열의 자료값은 다음과 같습니다.\n");
    printarray(x, rowsize, colsize);

    printf("2차원 배열 원소합은 %.3lf 입니다.\n",
                                    sum(x, rowsize, colsize));

    return 0;
}
```

함수로 2차원 배열 전달
- 배열의 이름만 전달

```c
double sum(double data[][2], int rowsize, int colsize)
{
    double total = 0;
    for (int i = 0; i < rowsize; i++)
        for (int j = 0; j < colsize; j++)
            total += data[i][j];

    return total;
}

void printarray(double data[][2], int rowsize, int colsize)
{
    for (int i = 0; i < rowsize; i++)
    {
        printf("% d행 원소: ", i + 1);
        for (int j = 0; j < colsize; j++)
            printf("x[%d][%d] = %5.2lf ", i, j, data[i][j]);
        printf("\n");
    }
    printf("\n");
}
```

```
2차원 배열의 자료값은 다음과 같습니다.
 1행 원소: x[0][0] = 11.00   x[0][1] = 12.00
 2행 원소: x[1][0] = 21.00   x[1][1] = 22.00
 3행 원소: x[2][0] = 31.00   x[2][1] = 32.00

2차원 배열 원소합은 129.000 입니다.
```

Questions?