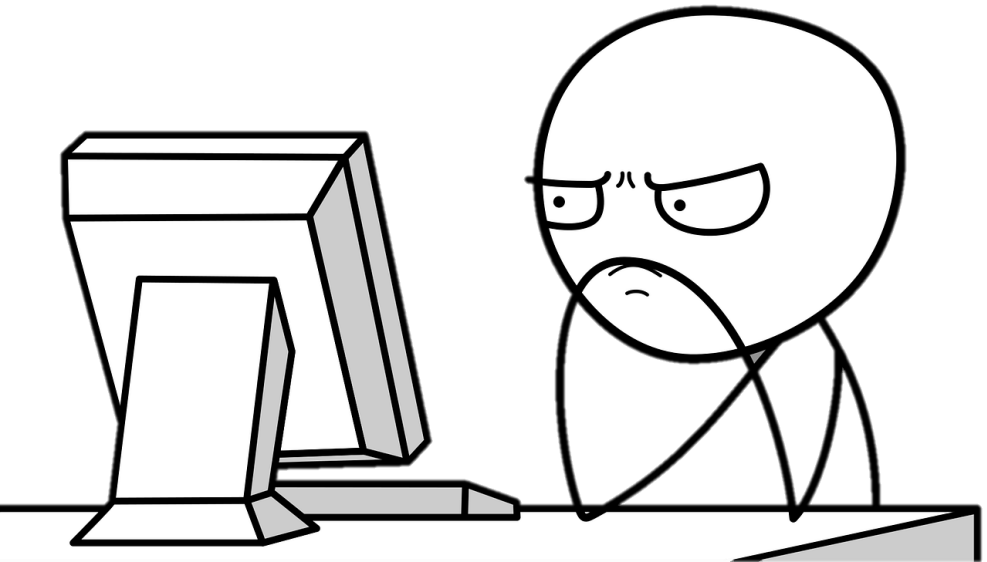

컴퓨터 프로그래밍
Computer Programming

10
함수 기초

목차

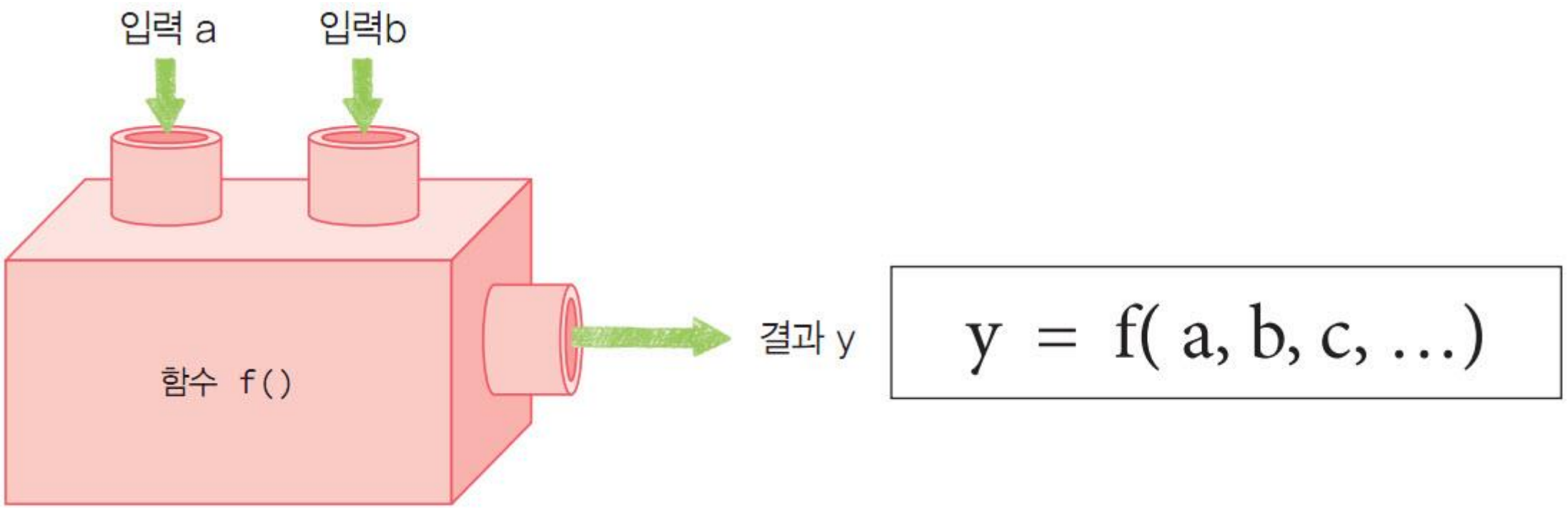
1. 함수정의와 호출



함수function

- 프로그램에서 특정한 작업을 처리하도록 작성한 프로그램 단위
- 필요한 입력을 받아 원하는 기능을 수행한 후 결과를 반환return하는 프로그램 단위
- 프로그램에서 원하는 특정한 작업을 수행하도록 설계된 독립된 프로그램 단위
- C 프로그램이란 여러 함수의 집합으로 구성되는 프로그램
- C 프로그램은 최소한 main() 함수와 다른 함수로 구성되는 프로그램

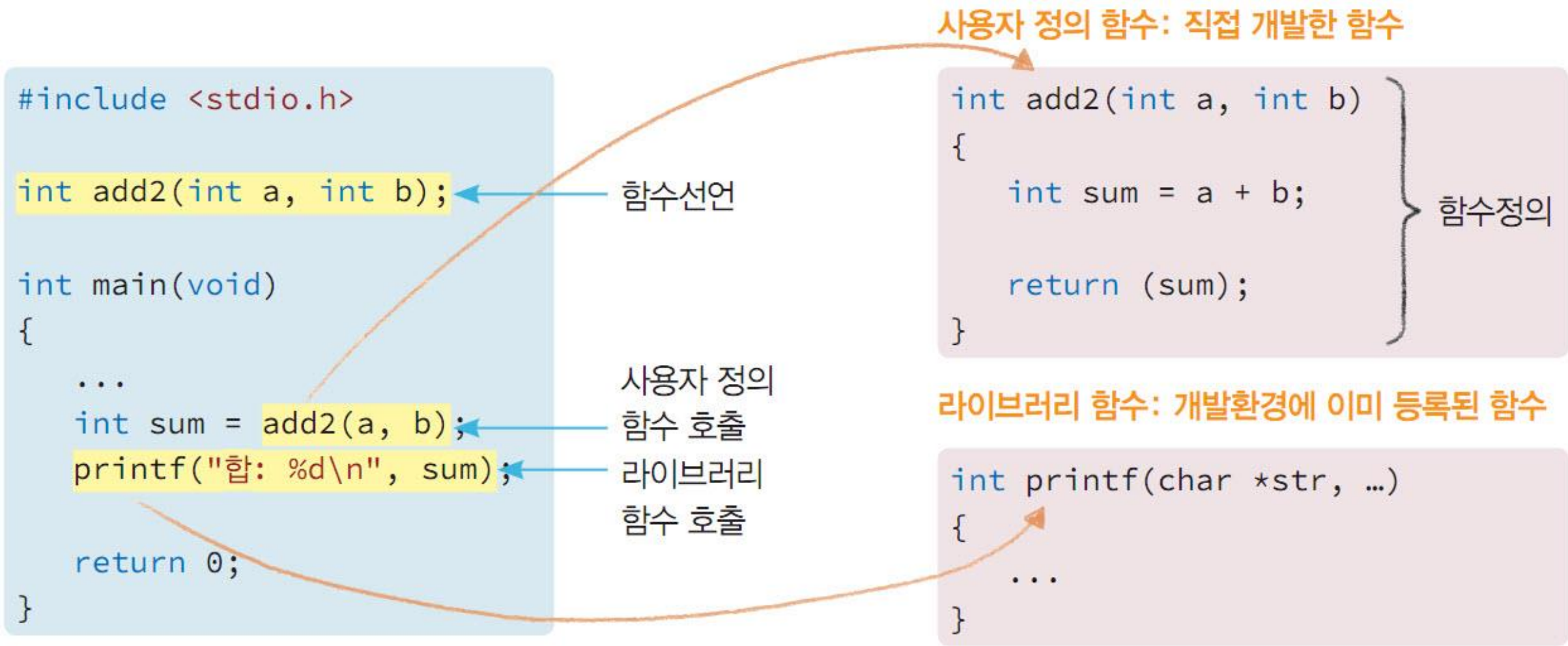
함수function



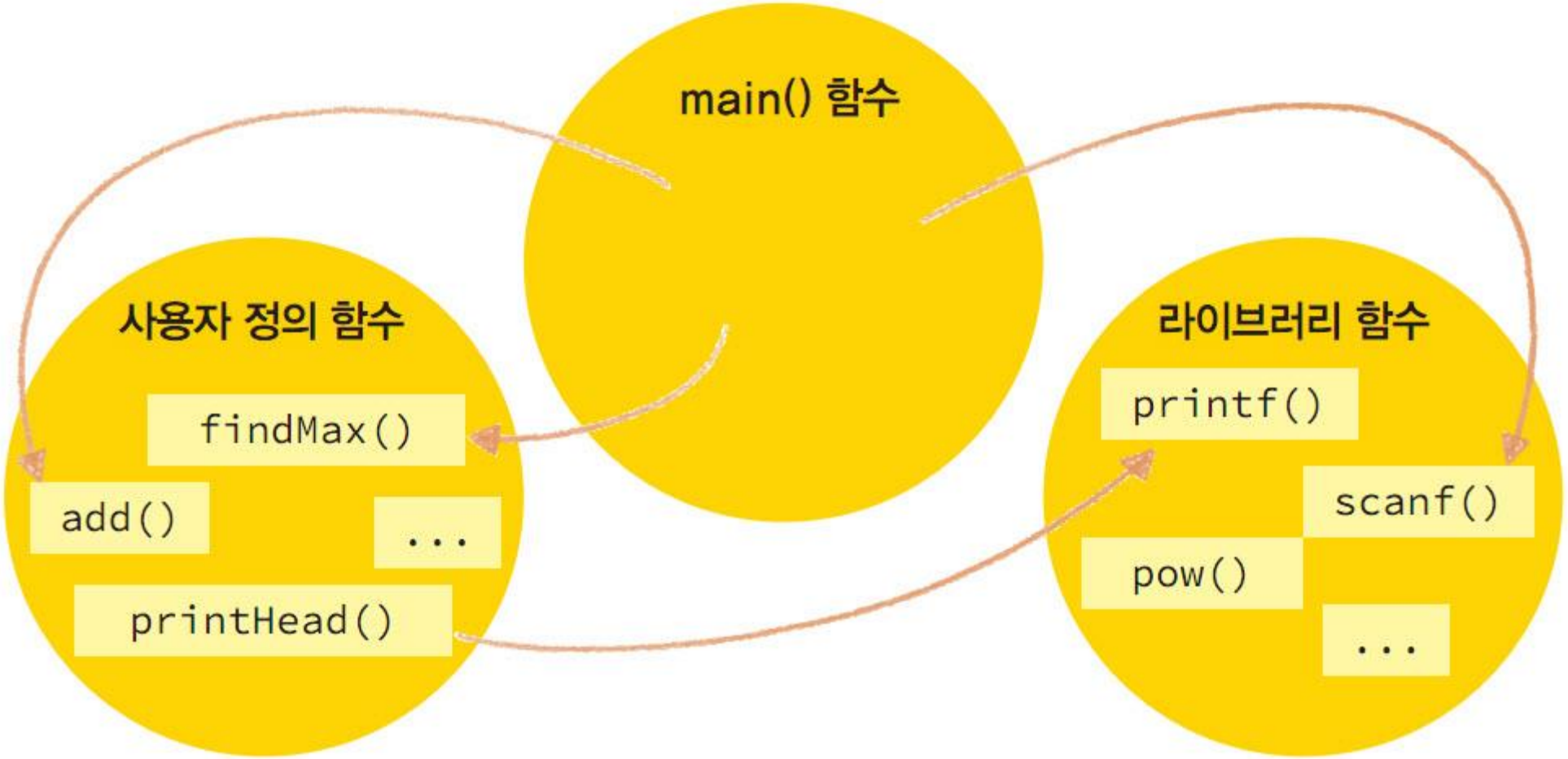
함수의 구분

- 사용자 정의 함수user defined function
  - 필요에 의해서 개발자가 직접 개발하는 함수
- 라이브러리 함수library function: 라이브러리library 또는 표준 함수standard function
  - printf()와 scanf()와 같이 이미 개발환경에 포함되어 있는 함수
  - 라이브러리 함수의 이용은 원하는 책을 도서관에서 대출 받아 이용하는 방식
- 프로그램은 라이브러리와 개발자가 만든 여러 함수로 구성
- main() 함수의 첫 줄에서 시작하여 마지막 줄을 마지막으로 실행한 후 종료
- 사용자가 직접 개발한 함수를 사용하기 위해서는 함수선언function declaration, 함수호출function call, 함수 정의function definition가 필요
- 필요에 따라 여러 소스 파일로 나누어 프로그래밍

함수의 선언, 호출, 정의


사용자 정의 함수: 직접 개발한 함수
#include <stdio.h>
int add2(int a, int b);
int main(void)
{
...
int sum = add2(a, b);
printf("합: %d\n", sum);
return 0;
}
함수선언
사용자 정의
함수 호출
라이브러리
함수 호출
int add2(int a, int b)
{
int sum = a + b;
return (sum);
}
함수정의
라이브러리 함수: 개발환경에 이미 등록된 함수
int printf(char *str, …)
{
...
}

절차적 프로그래밍


main() 함수
사용자 정의 함수
findMax()
add()
...
printHead()
라이브러리 함수
printf()
scanf()
pow()
...

함수정의function definition

- 함수머리function header와 함수몸체function body로 구성
- 함수머리function header
  - 반환형과 함수이름, 매개변수 목록으로 구성
  - 반환형: 함수 결과값의 자료형
  - 함수이름: 식별자의 생성규칙을 따름
  - 매개변수 목록: 자료형 변수이름의 쌍으로 필요한 수만큼 콤마로 구분하여 기술
- 함수몸체function body
  - {…}와 같이 중괄호로 시작하여 중괄호로 종료
  - 함수몸체에서는 함수가 수행해야 할 문장들로 구성
  - 함수몸체의 마지막은 대부분 결과값을 반환하는 return문장으로 종료

함수정의function definition

함수정의

함수 정의 구현

- 이름 add2로 두 정수의 합을 구하는 함수를 정의
  - 함수머리
    int add2(int a, int b)
  - 함수몸체
    {
      int sum = a + b;
      return (sum);
    }
  - 매개변수parameter variable
    (int a, int b)

함수 정의 구현

```
int add2(int a, int b)
{
    int sum = a + b;

    return (sum);
}
```

함수헤더

함수몸체

```
int findMax2(int a, int b)
{
    int max = a > b ? a : b;

    return max;
}
```

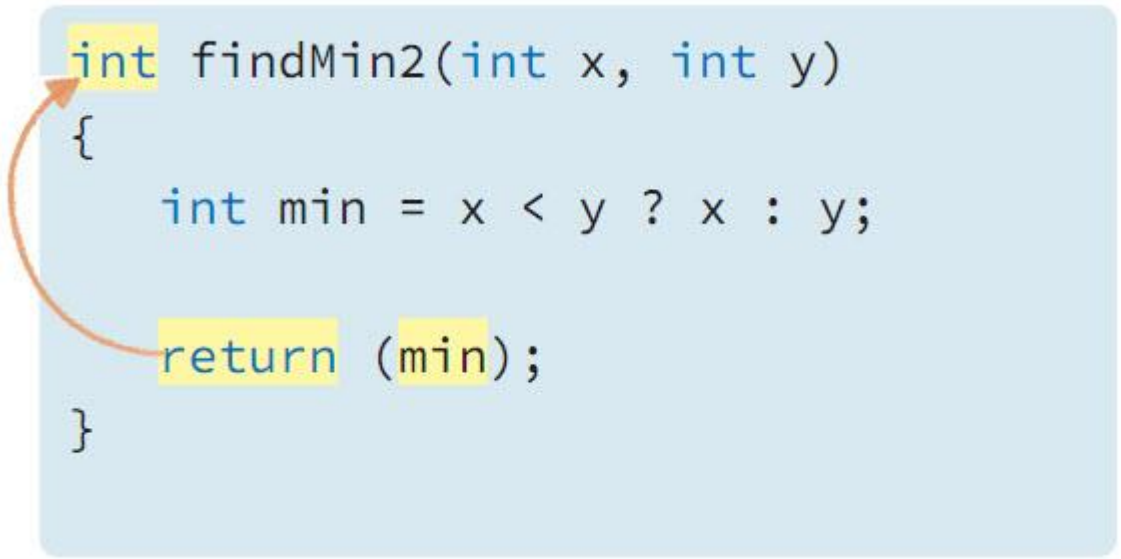
함수헤더

함수몸체

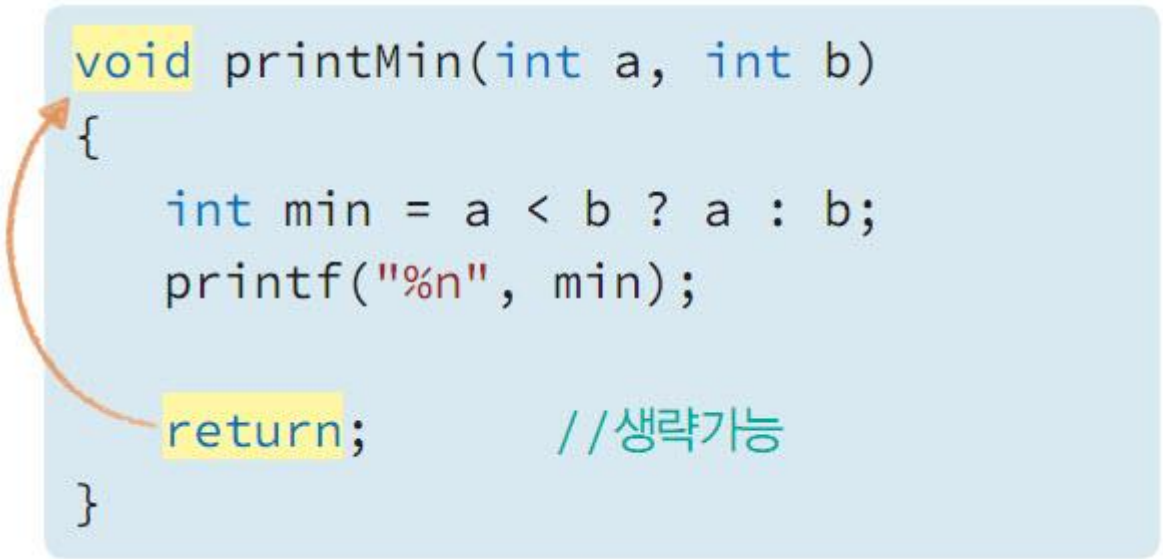
함수 정의 구현

- 이름 findMax2
  - 매개변수인 두 정수의 최대값을 구하는 함수를 정의
- 반환형과 return
  - char, short, int, long, float, double 등은 반환형이나 인자의 자료형으로도 이용 가능
  - 함수가 반환값이 없다면 반환형으로 void를 기술
    - 반환형을 아예 생략하는 것은 반환형을 int 임
    - 반환형을 생략하는 것은 적절하지 못한 방법, 반드시 기술
  - return 문장
    - 함수에서 반환값을 전달하는 목적과 함께 함수의 작업 종료를 알리는 문장

함수 정의 구현



```c
int findMin2(int x, int y)
{
    int min = x < y ? x : y;

    return (min);
}
```

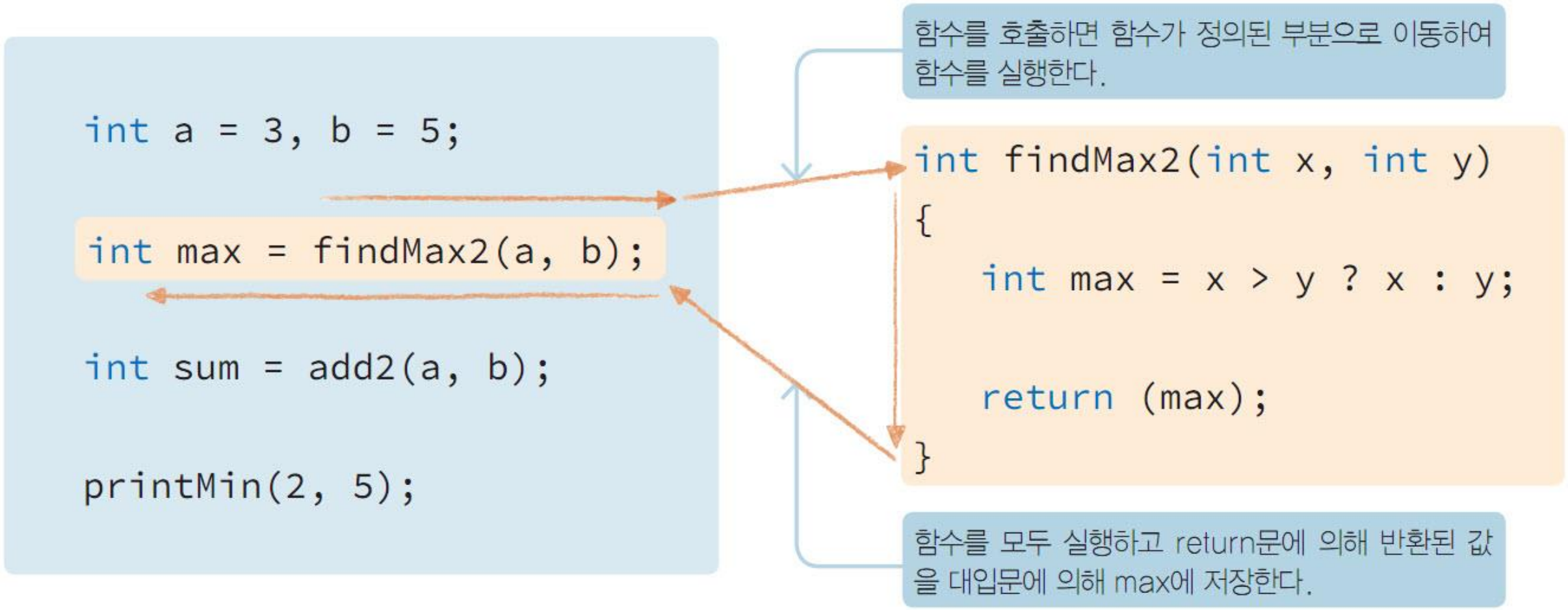


```c
void printMin(int a, int b)
{
    int min = a < b ? a : b;
    printf("%n", min);

    return;        //생략가능
}
```

함수 실행

- 프로그램 실행 중에 함수호출function call이 필요
- 함수를 호출하려면 함수이름과 함께 괄호 안에 적절한 매개변수가 필요
  - 즉 findMax2(a, b), add2(a, b), printMin(3, 5)와 같이 호출
- 함수 add2(a, b) 또는 findMax2(a, b)와 같이 반환값이 있는 함수
  - 반환된 값을 저장하기 위해 왼쪽에 변수(l-value)가 위치하는 대입연산자가 필요
  - 문장 sum = add2(5, 7);은 함수 반환값 12를 변수 sum에 저장하는 기능을 수행

함수 실행


int a = 3, b = 5;
int max = findMax2(a, b);
int sum = add2(a, b);
printMin(2, 5);
함수를 호출하면 함수가 정의된 부분으로 이동하여 함수를 실행한다.
int findMax2(int x, int y)
{
int max = x > y ? x : y;
return (max);
}
함수를 모두 실행하고 return문에 의해 반환된 값을 대입문에 의해 max에 저장한다.

함수 원형

- 함수 원형function prototype은 함수를 선언하는 문장
  - 함수원형 구문은 함수헤더에 세미콜론을 넣은 문장
  - int add2(int a, int b);
    - 매개변수의 변수이름은 생략 가능
    - int add2(int, int);도 가능

함수원형

반환형 함수이름(매개변수 목록);

함수원형은 문장이므로 마지막에 세미콜론 ;이 반드시 필요하다.

```
int add2(int a, int b);
int add2(int, int);
int findMin2(int x, int y);
int findMin2(int, int);
```

함수 원형

- 함수 원형function prototype은 함수를 선언하는 문장
  - 변수선언과 같이 함수를 호출하기 전에 반드시 선언
    - 함수의 사용을 미리 컴파일러에게 알리며 또한 프로그램을 이해하기 쉽게 도와주는 역할

함수원형을 함수 main() 위에 배치

```
#include <stdio.h>

int add2(int a, int b);          // 함수선언
//int add2(int, int) 도가능

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

   int sum = add2(a, b);         // 함수호출
   ...

   return 0;
}

int add2(int a, int b)           // 함수정의
{
   int sum = a + b;
   return (sum);
}
```

함수원형을 함수 main() 내부에 배치

```
#include <stdio.h>

int main(void)
{
   int add2(int a, int b);       // 함수선언
   //int add2(int, int) 도가능

   int a = 3, b = 5;

   int sum = add2(a, b);         // 함수호출
   ...

   return 0;
}

int add2(int a, int b)           // 함수정의
{
   int sum = a + b;
   return (sum);
}
```

Source Code #01: functionadd2.c

```c
// file: functionadd2.c
#include <stdio.h>

//int add2(int a, int b);   //이 위치도 가능

int main(void)
{
    int a = 3, b = 5;
    int add2(int a, int b); //int add2(int, int) 도가능

    //위 함수원형이 없으면 함수호출에서 오류발생
    int sum = add2(a, b);
    printf("합: %d\n", sum);

    return 0;
}

//함수 add2의 함수구현 또는 함수정의 부분
int add2(int a, int b)
{
    int sum = a + b;

    return (sum); //괄호는 생략 가능
}

//위 main() 함수에서 호출이 없으므로 이 함수구현은 실행되지 않음
int findMin2(int x, int y)
{
    int min = x < y ? x : y;

    return (min);
}
```

- 함수 add2()와 findMin2()를 구현한 예제
- 함수 findMin2()는 함수정의는 구현되었지만 함수호출이 없어 실행되지 않음
- 만일 함수 findMin2()를 실행하려면 main() 함수 내부에서 findMin2()를 호출

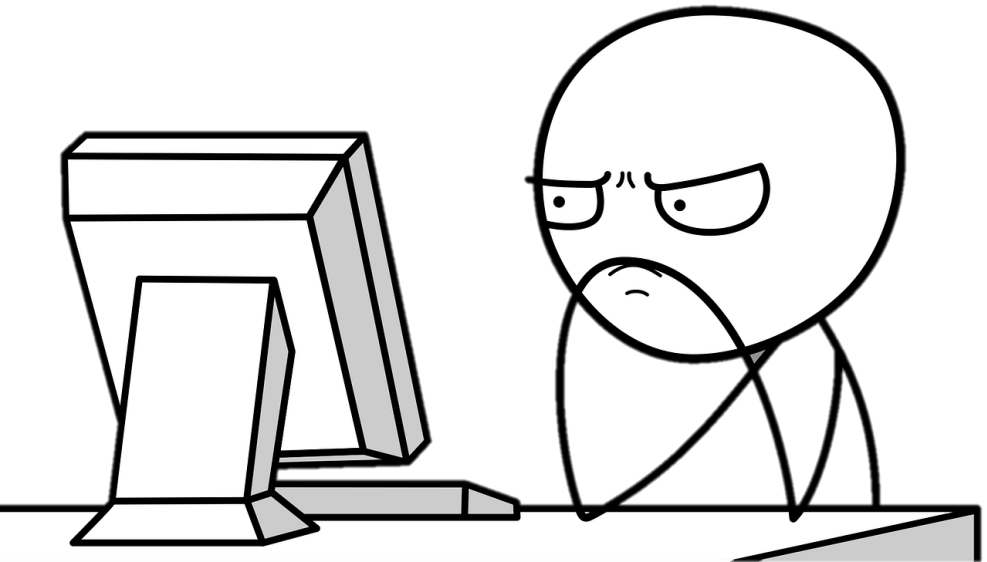
```
합: 8
```

Lab #01: 합을 구하는 함수 getsum()

- 함수 getsum(int n)
  - 1에서 매개변수인 n까지의 합을 구하는 함수
    - 표준입력으로 받은 양의 정수를 구현한 getsum()을 사용하여 적절히 호출하여 그 결과값을 출력
    - 함수 getsum()을 구현
    - 변수 max를 선언하여 표준입력으로 양의 정수를 입력
    - 함수 getsum()을 호출하여 1부터 max까지의 합을 출력
- 결과
  - 1에서 n까지의 합을 구할 n을 입력하시오. >> 100
  - 1에서 100까지의 합: 5050

```
1  // file: getsum.c
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  ____________________ //함수원형
6
7  int main(void)
8  {
9      int max = 0;
10
11     printf("1에서 n까지의 합을 구할 n을 입력하시오. >> ");
12     scanf("%d", &max);
13
14     printf("1에서 %d까지의 합: %d\n", ____________________); //함수호출
15
16     return 0;
17 }
18
19 int getsum(int n)
20 {
21     int sum = 0;
22     for (int i = 1; i <= n; i++)
23         ____________
24
25     return sum;
26 }
```

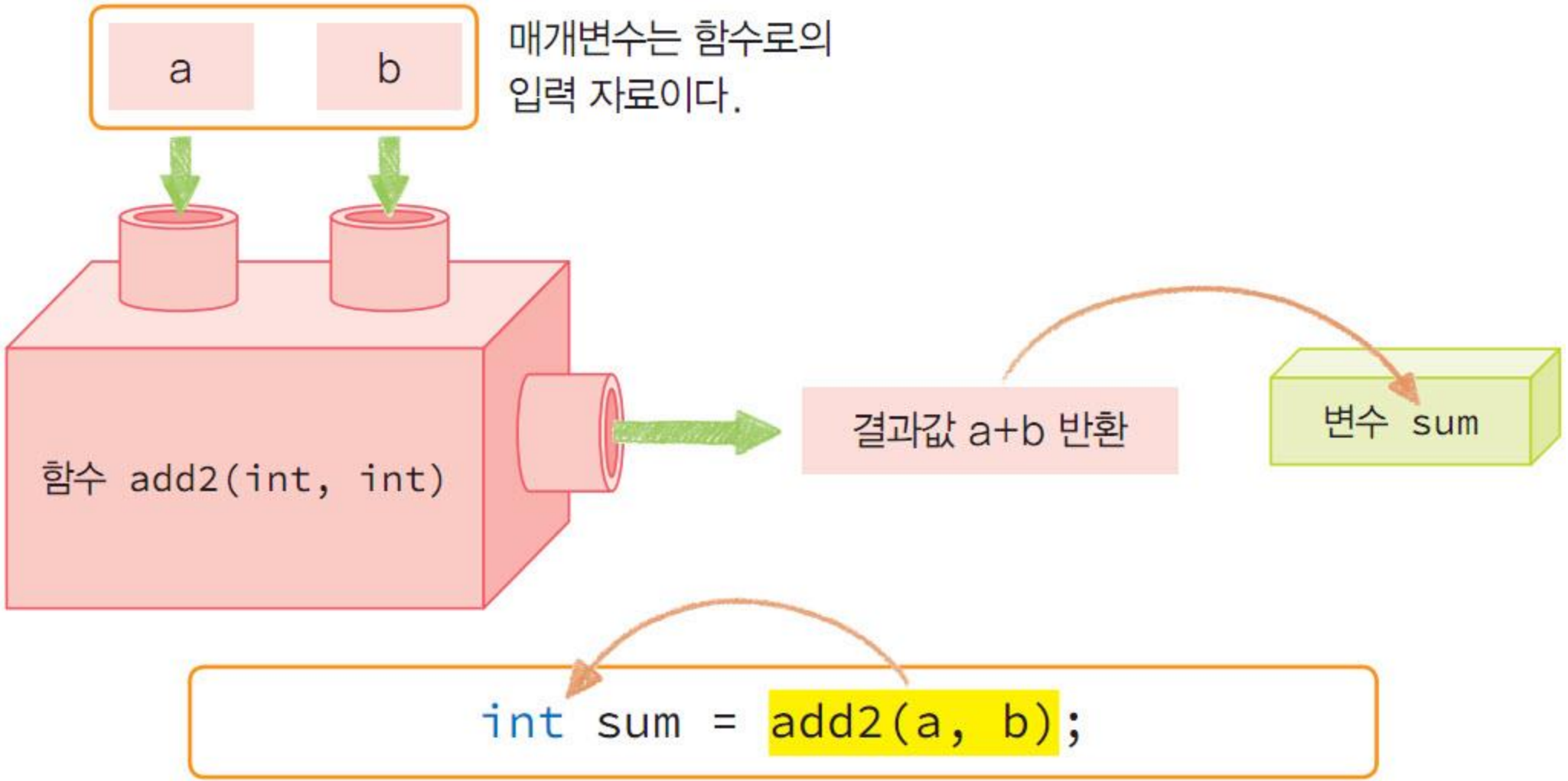
2. 함수의 매개변수 활용



매개변수와 인자

- 함수 매개변수parameter
  - 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용
  - 자료형과 변수명의 목록 표시: 여러 개 이용 가능
  - 필요 없으면 키워드 void 기술
  - 함수 매개변수와 반대로 함수를 호출하는 부분에서 함수가 작업을 수행
- 반환값을 이용
  - 다시 함수를 호출한 영역으로 결과값을 전달
  - 반환값은 하나만 이용할 수 있는 제약

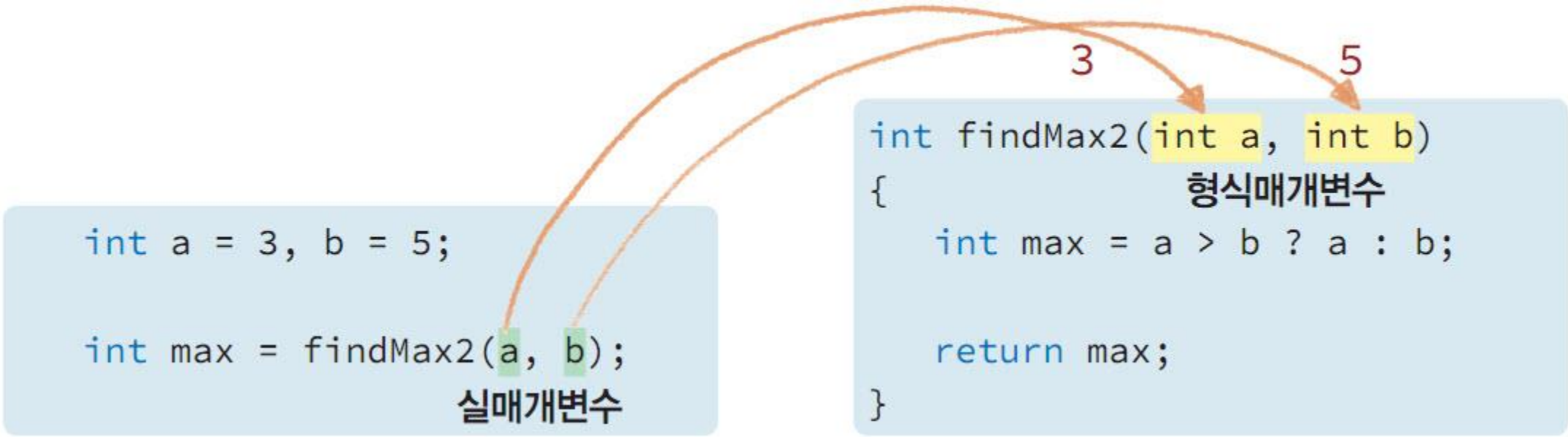
매개변수와 인자


a
b
매개변수는 함수로의
입력 자료이다.
함수 add2(int, int)
결과값 a+b 반환
변수 sum
int sum = add2(a, b);

형식매개변수와 실매개변수

- 형식매개변수formal parameters
  - 함수정의에서 기술되는 매개변수 목록의 변수
- 실매개변수real parameters
  - 함수를 호출할 때 기술되는 변수 또는 값
  - 실인자real argument 또는 인자argument라고도 부름
  - 실인자를 기술할 때는 함수헤더에 정의된 자료유형과 순서가 일치
  - 실인자의 수와 자료형이 다르면 문법오류가 발생
  - 형식인자의 변수는 그 함수가 호출되는 경우
  - 메모리에 할당allocation되고 함수가 종료, 메모리에서 자동으로 제거deallocation
  - 형식매개변수는 함수 내부 에서만 사용할 수 있는 변수

형식매개변수와 실매개변수



함수호출 시 매개변수의 수와 자료형이 다르면 문법오류가 발생한다.

```
int max = findMax2();          //오류발생
int max = findMax2(2);         //오류발생
int max = findMax2(2.4);       //오류발생
int max = findMax2(2.4, 6.8);  //오류발생
```

형식매개변수와 실매개변수

- 함수호출 findMax2(a, b)
  - 실인자인 a, b
  - 형식인자 a, b와는 다른 변수
- 값에 의한 호출call by value
  - 실인자의 값이 형식인자의 변수에 각각 복사된 후 함수가 실행
  - 매개변수의 값을 수정하더라도 함수를 호출한 곳에서의 실매개변수의 값은 변화되지 않는 특징

Source Code #02: functioncall.c

```c
// file: functioncall.c
#include <stdio.h>

int add2(int a, int b);     //int add2(int, int)도 가능
int findMax2(int, int);     //int findMax2(int a, int b)도 가능
void printMin(int, int);    //int printMin(int a, int b)도 가능

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

    int max = findMax2(a, b);
    printf("최대: %d\n", max);
    printf("합: %d\n", add2(a, b));

    //반환 값이 없는 함수호출은 일반문장처럼 사용
    printMin(2, 5);

    return 0;
}

//이하 함수 add2, findMax2, findMin2, printMin 구현
int add2(int a, int b)
{
    int sum = a + b;

    return (sum);
}

int findMax2(int a, int b)
{
    int max = a > b ? a : b;

    return max;
}

int findMin2(int x, int y)
{
    int min = x < y ? x : y;

    return (min);
}

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

    return;     //생략 가능
}
```

배열을 매개변수로 사용

- 배열이름으로 전달
  - 함수의 매개변수로 배열을 전달한다면 한 번에 여러 개의 변수를 전달하는 효과
- 함수 sum(): 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수
  - 형식매개변수: 실수형 배열 double ary[], 배열크기 int n
  - 첫 번째 형식매개변수에서 배열자체에 배열크기를 기술하는 것은 무의미
    - double ary[5]보다는 double ary[]라고 기술하는 것을 권장
  - 실제로 함수 내부에서 실인자로 전달되는 배열크기를 알 수 없음
    - 그러므로 배열크기를 두 번째 인자로 사용
    - 배열크기에 관계없이 배열 원소의 합을 구하는 함수를 만들려면 배열크기도 하나의 인자로 사용

배열을 매개변수로 사용

함수원형과 함수호출

```
double sum(double ary[], int n);
//double sum(double [], n); 가능

   ...

double data[] = {2.3, 3.4, 4.5, 6.7, 9.2};

   ... sum(data, 5);
```

함수정의

```
double sum(double ary[], int n)
{
   int i = 0;
   double total = 0.0;
   for (i = 0; i < n; i++)
      total += ary[i];

   return total;
}
```

함수호출 시 배열이름으로 배열인자를 명시한다.

다양한 배열원소 참조 방법(1)

- 배열 point
  - 간접연산자를 사용한 배열원소의 접근 방법은 *(point+i)
  - 배열의 합을 구하려면 sum += *(point + i); 문장을 반복
- 문장 int *address = point;
  - 이제 문장 sum += *(address++)으로도 배열의 합을 구할 수 있음
  - 그러나 point는 주소 상수라서 sum += *(point++)는 불가능
  - 증가연산식 point++의 피연산자로 상수인 point를 사용할 수 없기 때문

다양한 배열원소 참조 방법(1)

```
int i, sum = 0;
int point[] = {95, 88, 76, 54, 85, 33, 65, 78, 99, 82};
int *address = point;
int aryLength = sizeof (point) / sizeof (int);
```

가능

```
for (i=0; i<aryLength; i++)
  sum += *(point+i);
```

가능

```
for (i=0; i<aryLength; i++)
   sum += *(address++);
```

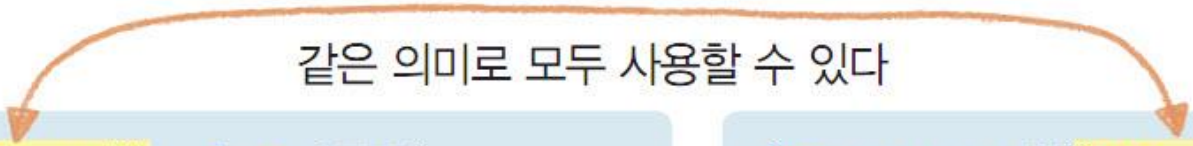
오류

```
for (i=0; i<aryLength; i++)
  sum += *(point++);
```

다양한 배열원소 참조 방법(2)

- 함수에서의 배열 전달
  - 함수헤더에 int ary[]로 기술하는 것은 int *ary로도 대체 가능
- for 문의 블록: 다음 4 개 문장 모두 가능
  - 변수 ary는 포인터 변수
  - 주소값을 저장하는 변수이므로 증가연산자의 이용 가능
  - 연산식 *ary++에서 후위 증가연산자 (ary++)의 우선순위가 가장 높기 때문에 *(ary++)와 같은 의미

다양한 배열원소 참조 방법(2)




```c
int sumary(int ary[], int SIZE)
{
    ...
}
```

```c
int sumaryf(int *ary, int SIZE)
{
    ...
}
```

```c
for (i = 0; i < SIZE; i++)
{
    sum += ary[i];
}
```

```c
for (i = 0; i < SIZE; i++)
{
    sum += *(ary + i);
}
```

```c
for (i = 0; i < SIZE; i++)
{
    sum += *ary++;
}
```

```c
for (i = 0; i < SIZE; i++)
{
    sum += *(ary++);
}
```

Source Code #03: arrayparam.c

```c
// file: arrayparam.c
#include <stdio.h>

//int sumaryf(int ary[], int SIZE);
int sumary(int *ary, int SIZE);

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

    int sum = 0;
    for (int i = 0; i < aryLength; i++)
        sum += *(point + i);
    //sum += *(point++);    //오류발생
    //sum += *(address++);  //가능

    printf("메인에서 구한 합은 %d\n", sum);
    address = point;
    printf("함수sumary() 호출로 구한 합은 %d\n", sumary(point, aryLength));
    printf("함수sumary() 호출로 구한 합은 %d\n", sumary(&point[0], aryLength));
    printf("함수sumary() 호출로 구한 합은 %d\n", sumary(address, aryLength));

    return 0;
}
```

```c

//int sumary(int ary[], int SIZE)도 가능
int sumary(int *ary, int SIZE)
{
    int sum = 0;

    for (int i = 0; i < SIZE; i++)
    {
        //sum += ary[i];        //가능
        //sum += *(ary + i);    //가능
        sum += *ary++;
        //sum += *(ary++);      //가능
    }

    return sum;
}
```

```
메인에서 구한 합은 755
함수sumary() 호출로 구한 합은 755
함수sumary() 호출로 구한 합은 755
함수sumary() 호출로 구한 합은 755
```

이차원 배열이름으로 전달

- 다차원 배열을 인자로 이용하는 경우,
  함수원형과 함수정의의 헤더에서 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술
- 함수 sum(): 인자인 이차원 배열값을 모두 더하는 함수
- 함수 printarray(): 인자인 이차원 배열값을 모두 출력하는 함수

이차원 배열이름으로 전달

함수원형과 함수호출

```
...
//이차원 배열값을 모두 더하는 함수원형
double sum(double data[][3], int, int);
//이차원 배열값을 모두 출력하는 함수원형
void printarray(double data[][3], int, int);

...
double x[][3] = { {1, 2, 3}, {7, 8, 9},
 {4, 5, 6}, {10, 11, 12} };

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

printarray(x, rowsize, colsize);

... sum(x, rowsize, colsize)

...
```

함수정의

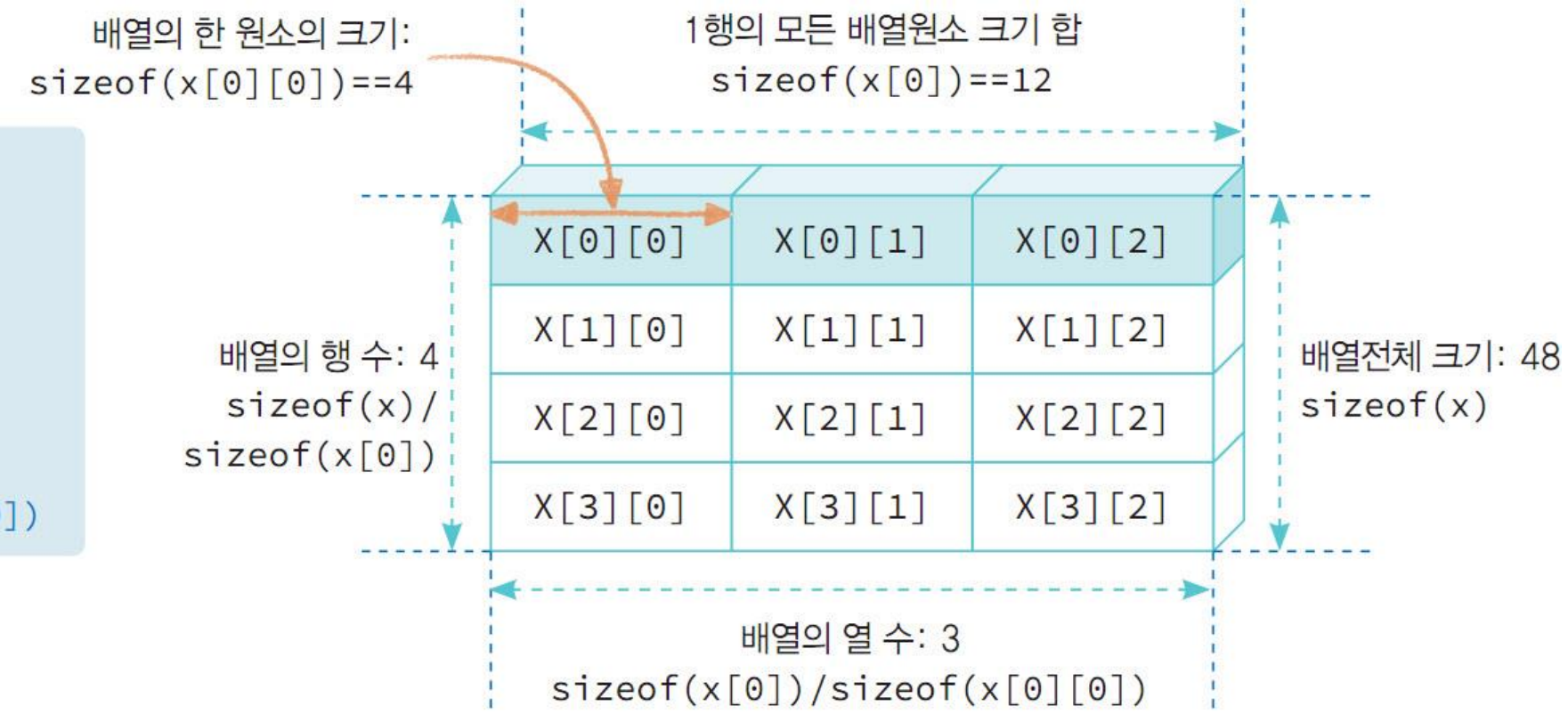
```
//이차원 배열값을 모두 출력하는 함수
void printarray(double data[][3], int rowsize, int colsize)
{

...

}

//이차원 배열값을 모두 더하는 함수
double sum(double data[][3], int rowsize, int colsize)
{
...
    for (i = 0; i < rowsize; i++)
        for (j = 0; j < colsize; j++)
            total += data[i][j];
    return total;
}
```

이차원 배열의 행과 열 수

- 함수 sum()을 호출
- 배열이름과 함께 행과 열의 수가 필요
- 이차원 배열의 행의 수: (sizeof(x) / sizeof(x[0]))로 계산
- 이차원 배열의 열의 수
  - (sizeof(x[0]) / sizeof(x[0][0]))로 계산
  - sizeof(x)는 배열 전체의 바이트 수
  - sizeof(x[0])는 1행의 바이트 수
  - sizeof(x[0][0])은 첫 번째 원소의 바이트 수

이차원 배열의 행과 열 수

```
배열의 전체 원소 수: 12
sizeof(x) / sizeof(x[0][0])

배열의 행 수: 4
sizeof(x) / sizeof(x[0])

배열의 열 수: 3
sizeof(x[0]) / sizeof(x[0][0])
```



Source Code #04: twoarrayfunction.c

```c
// file: twodarrayfunction.c
#include <stdio.h>

//2차원 배열값을 모두 더하는 함수원형
double sum(double data[][3], int, int);
//2차원 배열값을 모두 출력하는 함수원형
void printarray(double data[][3], int, int);

int main(void)
{
    //4 x 3 행렬을 위한 이차원 배열 선언 및 초기화
    double x[][3] = { { 1, 2, 3 }, { 7, 8, 9 }, { 4, 5, 6 }, { 10, 11, 12 } };

    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;
}
```

```
2차원 배열의 자료값은 다음과 같습니다.
 1행원소: x[0][0] =  1.00   x[0][1] =  2.00   x[0][2] =  3.00
 2행원소: x[1][0] =  7.00   x[1][1] =  8.00   x[1][2] =  9.00
 3행원소: x[2][0] =  4.00   x[2][1] =  5.00   x[2][2] =  6.00
 4행원소: x[3][0] = 10.00   x[3][1] = 11.00   x[3][2] = 12.00

2차원 배열 원소합은 78.000 입니다.
```

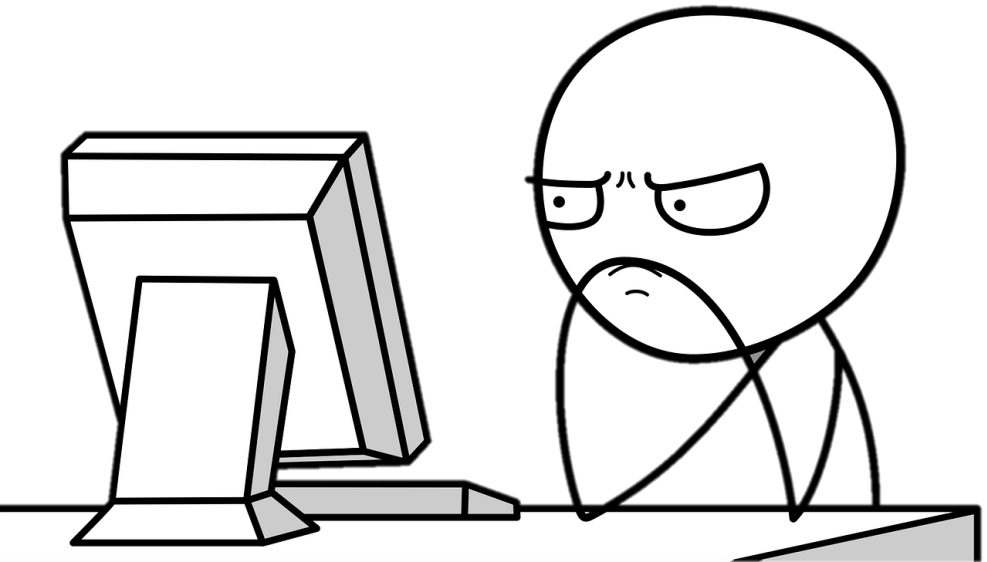
```c
//배열값을 모두 더하는 함수
double sum(double(*data)[3], int rowsize, int colsize)
//double sum(double data[][3], int rowsize, int colsize)
{
    double total = 0;

    for (int i = 0; i < rowsize; i++)
        for (int j = 0; j < colsize; j++)
            total += data[i][j];

    return total;
}

//배열값을 모두 출력하는 함수
void printarray(double(*data)[3], int rowsize, int colsize)
//void printarray(double data[][3], int rowsize, int colsize)
{
    for (int i = 0; i < rowsize; i++)
    {
        printf("% d행원소: ", i + 1);
        for (int j = 0; j < colsize; j++)
            printf("x[%d][%d] = %5.2lf   ", i, j, data[i][j]);
        printf("\n");
    }
    printf("\n");
}
```

Lab #02: 배열의 모든 원소를 지정한 만큼 증가시키는 함수

- 함수 incrementary(int ary[], int n, int SIZE)
  - 배열크기가 SIZE인 배열 ary의 모든 원소를 n만큼 증가시키는 함수
  - 초기화로 선언한 배열 data를 사용하여 모든 원소를 3만큼 증가시키는 함수 호출
    - 변수 aryLength 는 배열의 크기를 저장
    - 함수 prinary(int *data, int SIZE): 배열크기가 SIZE인 배열 ary의 모든 원소를 출력
- 결과
  - 47235
  - 배열 원소에 각각 3을 더한 결과:
  - 710568

```c
// file: incrementary.c
#include <stdio.h>

//함수 incrementary의 함수원형
[                                ]
//함수 printary의 함수원형
void printary(int *data, int SIZE);

int main(void)
{
    int data[] = { 4, 7, 2, 3, 5 };
    int aryLength = sizeof(data) / sizeof(int);

    //배열 출력을 위해 printary() 함수호출
    printary(data, aryLength);
    //배열 원소를 모두 3씩 증가시키기 위해 incrementary() 함수호출
    incrementary([                ]);
    printf("배열 원소에 각각 3의 더한 결과: \n");
    //결과를 알아 보기 printary() 함수호출
    printary(data, aryLength);

    return 0;
}

//배열크기가 SIZE인 배열 ary의 모든 원소를 n만큼 증가시키는 함수
void incrementary(int ary[], int n, int SIZE)
{
    for (int i = 0; i < SIZE; i++)
        [                                ]
}

//배열크기가 SIZE인 배열 ary의 모든 원소를 출력하는 함수
void printary(int *data, int SIZE)
{
    for (int i = 0; i < SIZE; i++)
        printf("%2d ", data[i]);
    printf("\n");
}
```

3. 재귀와 라이브러리 함수



재귀 함수recursive function

- 함수구현에서 자신 함수를 호출하는 함수
- 재귀적 특성을 표현하는 알고리즘
- 재귀 함수를 이용하면 문제를 쉽게 해결할 수 있고 이해하기도 쉬움

$$n! \begin{cases} 0! = 1 \\ \\ n! = n * (n-1)! \quad \text{for } (n >= 1) \end{cases}$$

수학 수식에서 재귀적 특성

- n 계승n factorial을 나타내는 수식 n!
  - 1 * 2 * 3 * … * (n-2) * (n-1) * n을 의미
  - 즉 n!의 정의에서 보듯 계승은 재귀적 특성
- n!을 구하기 위해서 (n-1)!을 먼저 구한다면 쉽게 n!을 구할 수 있음
  - (n-1)!을 구하기 위해서는 다시 (n-2)!이 필요
- n!을 함수 factorial(n)로 구현한다면 함수 factorial(n) 구현에서 다시 factorial(n-1)을 호출하여 그 결과를 이용 가능

n!을 위한 함수 factorial()

```
if   (n <= 1)
       n! = 1
else
       n! = n * (n-1)!
```



```
int factorial(int num)
{
    if (num <= 1)
        return 1;
    else
        return (num * factorial(num - 1));
}
```

Source Code #05: factorial.c

```c
// file: factorial.c
#include <stdio.h>

int factorial(int); //함수원형

int main(void)
{
    for (int i = 1; i <= 10; i++)
        printf("%2d! = %d\n", i, factorial(i));

    return 0;
}

// n! 구하는 재귀함수
int factorial(int number)
{
    if (number <= 1)
        return 1;
    else
        return (number * factorial(number - 1));
}
```

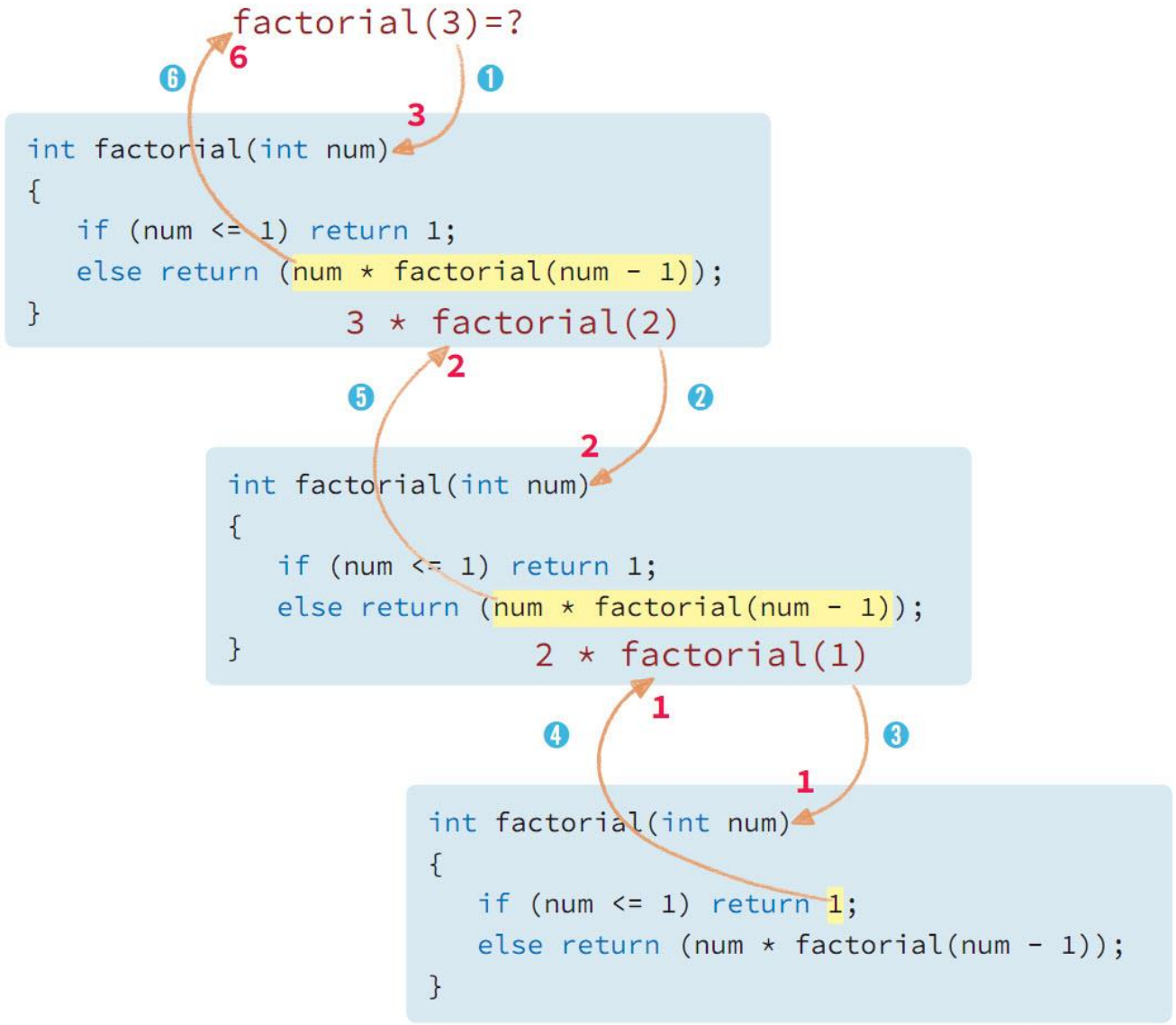
- 재귀함수 factorial()을 이용하여 1!에서 10!까지 결과를 출력

```
 1! = 1
 2! = 2
 3! = 6
 4! = 24
 5! = 120
 6! = 720
 7! = 5040
 8! = 40320
 9! = 362880
10! = 3628800
```

재귀 함수의 실행

- 함수 factorial(n)에서 factorial(3)을 호출한 경우 실행과정
  - factorial(3)을 호출하면 함수 factorial(3) 내부에서 다시 factorial(2)를 호출
  - factorial(2)에서는 다시 factorial(1)을 호출
  - 결국 factorial(1) 내부에서 return 1을 실행
  - 다시 factorial(2)로 돌아와 반환값 1을 이용하여 return (2*1)을 실행
  - 계속해서 factorial(3)으로 돌아와 factorial(2)의 결과값인 2를 이용
  - return (3*2)를 실행하면 결국 6을 반환
- 재귀함수 장단점
  - 일반적으로 재귀함수는 함수의 호출이 계속되면 시간도 오래 걸리고 메모리의 사용도 많다는 단점

재귀 함수의 실행


factorial(3)=?
6
1
3
int factorial(int num)
{
if (num <= 1) return 1;
else return (num * factorial(num - 1));
}
3 * factorial(2)
2
5
2
2
int factorial(int num)
{
if (num <= 1) return 1;
else return (num * factorial(num - 1));
}
2 * factorial(1)
1
4
3
1
int factorial(int num)
{
if (num <= 1) return 1;
else return (num * factorial(num - 1));
}

난수random number 라이브러리 함수

- 함수 rand()
  - 특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수를 난수
- 함수 rand()의 함수원형은 헤더파일 stdlib.h에 정의
  - 함수 rand()를 사용하려면 헤더파일 stdlib.h를 추가
  - Visual C++에서는 함수 rand()
    - 0에서 32767사이의 정수 중에서 임의로 하나의 정수를 반환
  - Visual C++의 헤더 파일 stdlib.h 파일
    - 함수 rand()의 최대값으로 기호 상수 RAND_MAX를 16진수로 0x7fff로 정의
  - 임의의 수
    - 어느 수가 반환될 지 예측할 수 없으며 어느 수가 선택될 확률이 모두 동일하다는 의미

난수random number 라이브러리 함수

```
#include <stdlib.h> //rand() 위한 헤더파일 포함

int main(void)
{

    ...
      printf("%5d ", rand());

}
```

함수 rand()를 사용하려면 헤더 파일 stdlib.h를 삽입해야 한다.

함수 rand()는 0에서 32767 까지의 정수 중에서 하나를 반환한다

Source Code #06: rand.c

```c
// file: rand.c
#include <stdio.h>

#include <stdlib.h> //rand()위한 헤더파일 포함

int main(void)
{
    printf("0 ~ %5d 사이의 난수 5개: rand()\n", RAND_MAX);
    for (int i = 0; i < 5; i++)
        printf("%5d ", rand());
    puts("");

    return 0;
}
```

- 난수 5개 출력

```
0 ~ 32767 사이의 난수 5개: rand()
   41 18467  6334 26500 19169
```

함수 srand()

- 함수 rand()는 함수호출 순서에 따라 항상 일정한 수가 반환
  - 항상 41, 18467, 6334,26500, 19169 값들이 출력
- 매번 난수를 다르게 생성
  - 시드(seed)값: 시드값이란 난수를 다르게 만들기 위해 처음에 지정하는 수
  - 시드값이 다르면 난수가 달라짐
    - 함수 srand(seed)를 호출
    - 함수 rand()에서 호출
  - 서로 다른 seed 값을 지정하기 위해 함수 time() 이용
    - time(NULL)은 1970년 1월 1일 이후 현재까지 경과된 시간을 초 단위로 반환
    - 헤더 파일 time.h를 삽입
  - 1에서 n까지의 난수를 발생: rand() % n + 1을 이용
  - 일반화시켜 a에서 b까지의 난수를 발생: rand() % (b – a + 1) + a을 이용

함수 srand()

```c
#include <stdlib.h> //rand(), srand()를 위한 헤더파일 포함
#include <time.h>   //time()을 위한 헤더파일 포함

#define MAX 100

int main(void)
{
    ...
    srand((long) time(NULL));
    ...
        number = rand()%MAX + 1;
    ...
}
```

time(NULL) 값을 인자로 srand()를 호출한다.

1에서 MAX까지 하나의 난수가 생성된다.

Source Code #07: srand.c

```c
// file: srand.c
#include <stdio.h>
#include <stdlib.h> //rand(), srand()를 위한 헤더파일 포함

#include <time.h>   //time()을 위한 헤더파일 포함

#define MAX 100

int main(void)
{
    long seconds = (long)time(NULL);
    srand(seconds); //printf("%ld\n", seconds);

    printf("1 ~ %5d 사이의 난수 5개:\n", MAX);
    for (int i = 0; i < 5; i++)
        printf("%5d ", rand() % MAX + 1);
    puts("");

    return 0;
}
```

- 시드 지정 후 난수 출력

```
1 ~   100 사이의 난수 5개:
   97    90    45    34    38
```

Source Code #08: math.c

```c
// file: math.c
#include <stdio.h>

#include <math.h> //수학 관련 다양한 함수헤더 포함 헤더파일

int main(void)
{
    printf("  i   i제곱   i세제곱  제곱근(sqrt)\n");
    printf("-----------------------------------\n");
    for (int i = 3; i < 7; i++)
        printf("%3d %7.1f %9.1f %9.1f\n", i, pow(i, 2), pow(i, 3), sqrt(i));
    printf("\n");

    printf("exp(1.0) == %5.2f, ", exp(1.0));
    printf("pow(2.72, 1.0) == %5.2f, ", pow(2.72, 1.0));
    printf("sqrt(49) == %5.2f\n", sqrt(49));
    printf("abs(-10) == %5d, ", abs(-10));
    printf("ceil(7.1) == %5.2f, ", ceil(7.1));
    printf("floor(6.9) == %5.2f\n", floor(6.9));

    return 0;
}
```

- 수학 관련 함수를 사용하려면 헤더파일 math.h을 삽입
- 헤더파일 math.h에 선언되어 있는 수학 관련 함수

```
  i   i제곱   i세제곱  제곱근(sqrt)
-----------------------------------
  3     9.0      27.0       1.7
  4    16.0      64.0       2.0
  5    25.0     125.0       2.2
  6    36.0     216.0       2.4

exp(1.0) ==  2.72, pow(2.72, 1.0) ==  2.72, sqrt(49) ==  7.00
abs(-10) ==    10, ceil(7.1) ==  8.00, floor(6.9) ==  6.00
```

수학 관련 함수

함수	처리 작업
double sin(double x)	삼각함수 sin
double cos(double x)	삼각함수 cos
double tan(double x)	삼각함수 tan
double sqrt(double x)	제곱근, square root(x)
double exp(double x)	e^x
double log(double x)	$\log_e(x)$
double log10(double x)	$\log_{10}(x)$
double pow(double x, double y)	x^y
double ceil(double x)	x보다 작지 않은 가장 작은 정수
double floor(double x)	x보다 크지 않은 가장 큰 정수
int abs(int x)	정수 x의 절대 값
double fabs(double x)	실수 x의 절대 값

헤더파일 ctype.h

- 문자 관련 함수
  - 헤더파일 ctype.h에 매크로로 정의
  - 문자를 검사하거나 문자를 변환하는 기능을 수행
  - 일반적으로 검사 함수는 isxxx(char)로, 변환 함수는 toxxx(char)로 명명
    - 검사 함수는 0(false)과 0이 아닌 정수값(true)을 반환
    - 변환 함수는 변환된 문자를 반환
- 주요 함수
  - toupper(c): c가 영문 소문자일 때 영문 대문자로 변환
  - tolower(c): 각각 c가 영문 대문자일 때 영문 소문자로 변환

문자 관련 함수 매크로

함수 원형	기능
isalpha(char)	영문자 검사
isupper(char)	영문 대문자 검사
islower(char)	영문 소문자 검사
isdigit(char)	숫자(0~9) 검사
isxdigit(char)	16진수 숫자(0~9, A~F, a~f) 검사
isspace(char)	공백(' ','\n','\t','\f','\v','\r') 문자 검사
ispunct(char)	구두(빈칸이나 알파뉴메릭을 제외한 출력문자) 문자 검사
isalnum(char)	영문과 숫자(alphanumeric)(0~9, A~Z, a~z) 검사
isprint(char)	출력 가능 검사
isgraph(char)	그래픽 문자 검사
iscntrl(char)	제어문자('\a', '\b', '\n', '\t','\f','\v','\r') 검사
toupper(char)	영문 소문자를 대문자로 변환
tolower(char)	영문 대문자를 소문자로 변환
toascii(char)	아스키 코드로 변환
_tolower(char)	무조건 영문 소문자로 변환
_toupper(char)	무조건 영문 대문자로 변환

Source Code #09: char.c

```c
// file: char.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
#include <stdio.h>

#include <ctype.h> //문자 관련 함수는 헤더파일 ctype.h에 매크로로 정의

void print2char(char);

int main(void)
{
    char ch;

    printf("알파벳(종료x) 또는 다른 문자 입력하세요.\n");
    do
    {
        printf("문자 입력 후 Enter: ");
        scanf("%c", &ch);
        getchar();      //enter 키 입력 받음
        if (isalpha(ch))
            print2char(ch);
        else
            printf("입력: %c\n", ch);
    } while (ch != 'x' && ch != 'X'); //입력이 x 또는 X이면 종료

    return 0;
}

void print2char(char ch)
{
    if (isupper(ch))
        printf("입력: %c, 변환: %c\n", ch, tolower(ch));
    else
        printf("입력: %c, 변환: %c\n", ch, toupper(ch));

    return;
}
```

- 문자를 입력 받아 알파벳 문자이면 입력한 문자와 대소문자를 변환한 문자를 출력하는 프로그램
  - 문자 입력은 계속 받으며 센티널 값으로 문자 x를 입력 받으면 결과를 출력
  - 알파벳이 아닌 다른 문자를 입력 받으면 그대로 문자를 출력

다양한 헤더 파일

- 여러 라이브러리 함수를 위한 함수원형과 상수, 그리고 매크로가 존재

헤더파일	처리 작업
stdio.h	표준 입출력 작업
math.h	수학 관련 작업
string.h	문자열 작업
time.h	시간 작업
ctype.h	문자 관련 작업
stdlib.h	여러 유틸리티(텍스트를 수로 변환 등) 함수

Lab #03: 1에서 100 사이의 난수 알아 맞추기

- 가장 먼저 난수를 생성시켜 변수 guess에 저장
- 정수를 입력 받아 변수 input에 저장한 후
- 저장된 guess와 비교하여 틀렸으면 사용자에게 다음 입력을 위한 정보를 제공, 다시 반복
- 입력한 정수 input과 guess가 같으면 "정답입니다"를 출력하고 종료
- 함수 time()을 이용하기 위해 헤더 파일 time.h를 삽입
- 난수에 시드를 지정하기 위해 함수 srand((long) time(NULL))을 호출
- 1에서 n까지의 난수를 발생: 함수 rand()를 이용하여 수식 rand() % n + 1을 이용

```c
// file: numberguess.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#include <stdlib.h> //rand(), srand()를 위한 헤더파일 포함
#include <time.h>   //time()을 위한 헤더파일 포함

#define MAX 100

int main(void)
{
    int guess, input;

    srand((long)time(NULL));
    guess = ;

    printf("1에서 %d 사이에서 한 정수가 결정되었습니다.\n", MAX);
    printf("이 정수는 무엇일까요? 입력해 보세요. : ");

    while (scanf()) {
        if (input > guess)
            printf("입력한 수보다 작습니다. 다시 입력하세요. : ");
        else if (input < guess)
            printf("입력한 수보다 큽니다. 다시 입력하세요. : ");
        else
        {
            puts("정답입니다.");

        }
    }

    return 0;
}
```


I
PROGRAMMING