제10장

함수 기초

Perfect C

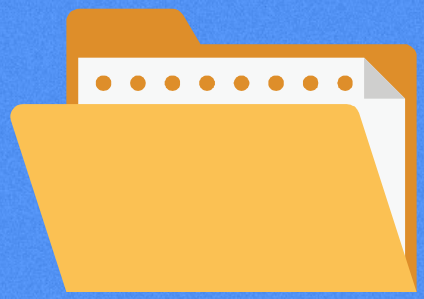
C언어로 배우는 프로그래밍 기초

단원 목표

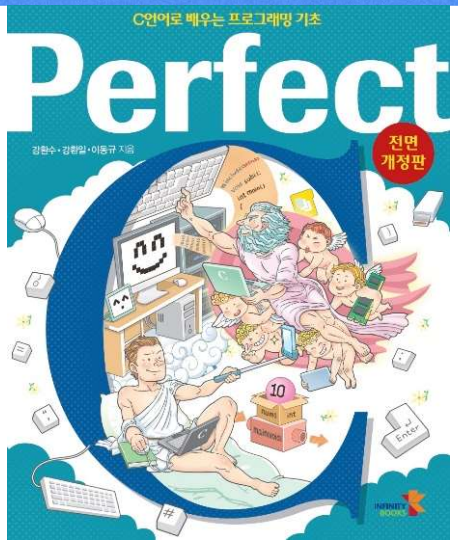
학습목표

- **함수에 대해 이해하고 설명할 수 있다.**
  - 함수의 개념, 함수의 입력과 반환
  - 라이브러리와 사용자 정의 함수
- **함수를 사용하기 위한 함수정의, 함수호출, 함수원형을 이해하고 설명할 수 있다.**
  - 함수 정의에서의 함수머리와 함수몸체
  - 함수호출 방법 및 실행 순서
  - 함수원형 구문과 필요성
  - 매개변수의 사용과 가변인자 구현 방법
- **재귀함수의 정의와 구현방법을 이해하고 설명할 수 있다.**
  - n!에서 재귀 특성을 파악하여 구현
  - 재귀함수의 실행과 장단점
- **라이브러리 함수를 이해하고 기본적인 라이브러리를 사용할 수 있다.**
  - 난수를 위한 함수 rand()의 이용과 시드값을 위한 srand()의 사용
  - 문자와 수학 관련 함수의 이용

학습목차



01. 함수정의와 호출



함수의 개념

- **프로그램에서**
  - 매번 되풀이하는 일정한 작업을 반복적으로 처리하는 일이 흔히 발생
- **함수(function)**
  - 프로그램에서 특정한 작업을 처리하도록 작성한 프로그램 단위
  - 필요한 입력을 받아 원하는 기능을 수행한 후
    - 결과를 반환(return)하는 프로그램 단위
  - 프로그램에서 원하는 특정한 작업을 수행하도록 설계된 독립된 프로그램 단위

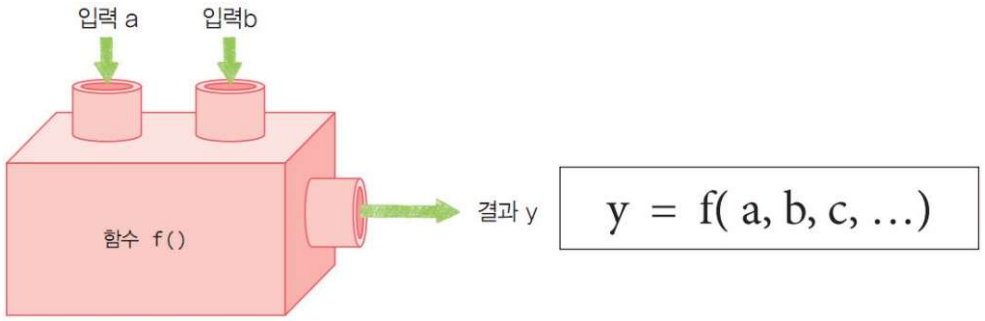


그림 10-1 함수 개념

  - C 프로그램이란 여러 함수의 집합으로 구성되는 프로그램
    - C 프로그램은 최소한 main() 함수와 다른 함수로 구성되는 프로그램

라이브러리와 사용자 정의 함수

- **함수의 구분**
  - 사용자 정의 함수(user defined function)
    - 필요에 의해서 개발자가 직접 개발하는 함수
  - 라이브러리 함수 (library function): 라이브러리(library) 또는 표준 함수(standard function)
    - printf()와 scanf()와 같이 이미 개발환경에 포함되어 있는 함수
    - 라이브러리 함수의 이용은 원하는 책을 도서관에서 대출 받아 이용하는 방식
- **하나의 프로그램**
  - 라이브러리와 개발자가 만든 여러 함수로 구성
  - main() 함수의 첫 줄에서 시작하여 마지막 줄을 마지막으로 실행한 후 종료
- **사용자가 직접 개발한 함수를 사용하기 위해서는**
  - 함수선언(function declaration), 함수호출(function call), 함수 정의(function definition)가 필요
  - 필요에 따라 여러 소스 파일로 나누어 프로그래밍

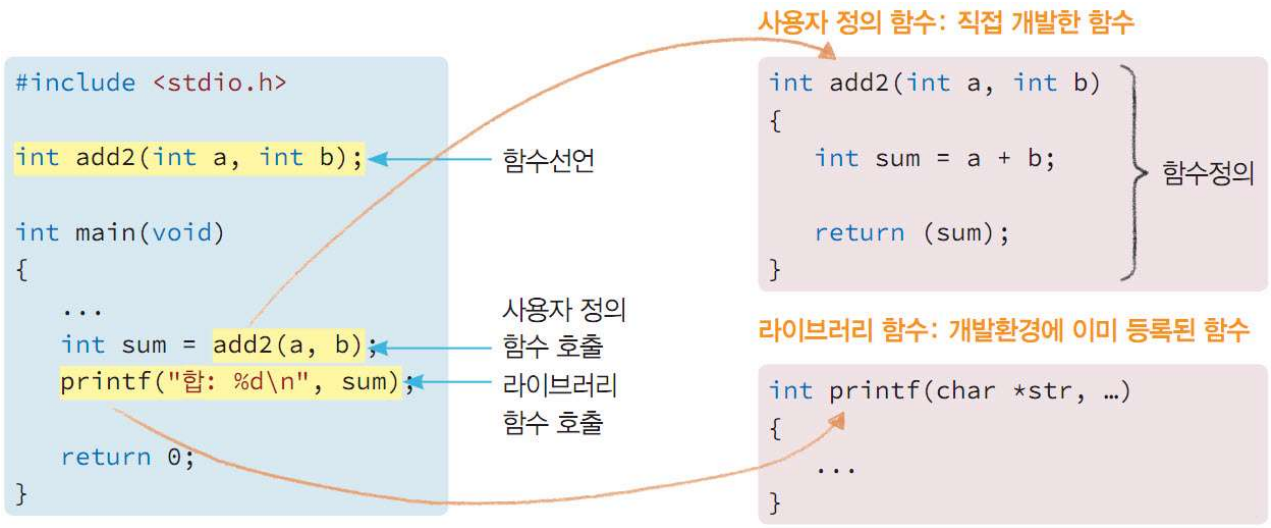


그림 10-2 함수선언, 함수호출, 함수정의

절차적 프로그래밍

- **여러 함수로 나누어 프로그래밍**
  - 주어진 문제를 여러 개의 작은 문제로 나누어 해결
  - 나뉘어진 각각의 함수 단위로 구현하면 대규모 프로그램도 개발 가능
- **모듈화 프로그램(modular program)**
  - 구조화된 프로그램(structured program)
    - 적절한 함수로 잘 구성된 프로그램
  - 한번 정의된 함수는 여러 번 호출이 가능
    - 소스의 중복을 최소화하여 프로그램의 양을 줄이는 효과
  - 잘 구현된 함수
    - 라이브러리와 같이 여러 프로그램에서 손쉽게 이용이 가능하며 유지관리도 편리
- **절차적 프로그래밍 (procedural programming) 방식**
  - 함수 중심의 프로그래밍 방식

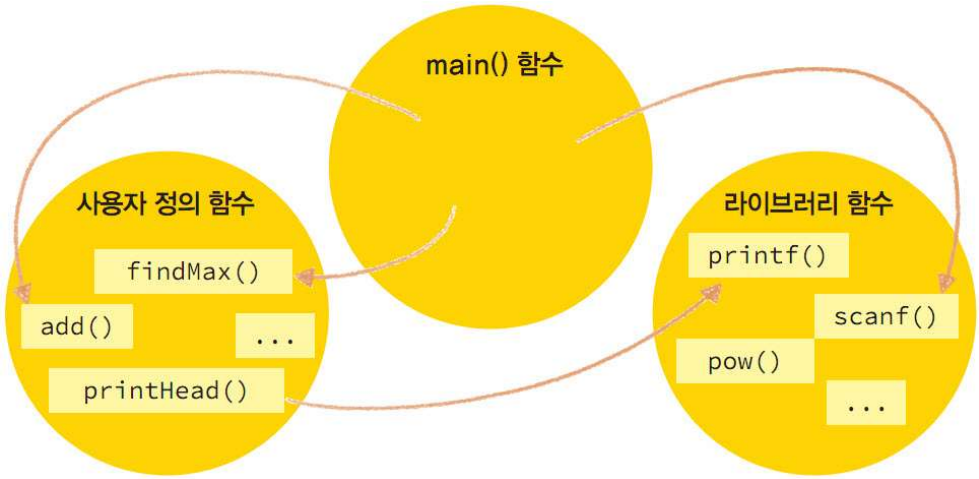


그림 10-3 절차적 프로그래밍

함수정의(function definition)

- **함수머리(function header)와 함수몸체(function body)로 구성**
  - 함수머리(function header)
    - 반환형과 함수이름, 매개변수 목록으로 구성
    - 반환형: 함수 결과값의 자료형
    - 함수이름: 식별자의 생성규칙을 따름
    - 매개변수 목록: 자료형 변수이름의 쌍으로 필요한 수만큼 콤마로 구분하여 기술
  - 함수몸체(function body)
    - {...}와 같이 중괄호로 시작하여 중괄호로 종료
    - 함수몸체에서는 함수가 수행해야 할 문장들로 구성
    - 함수몸체의 마지막은 대부분 결과값을 반환하는 return문장으로 종료

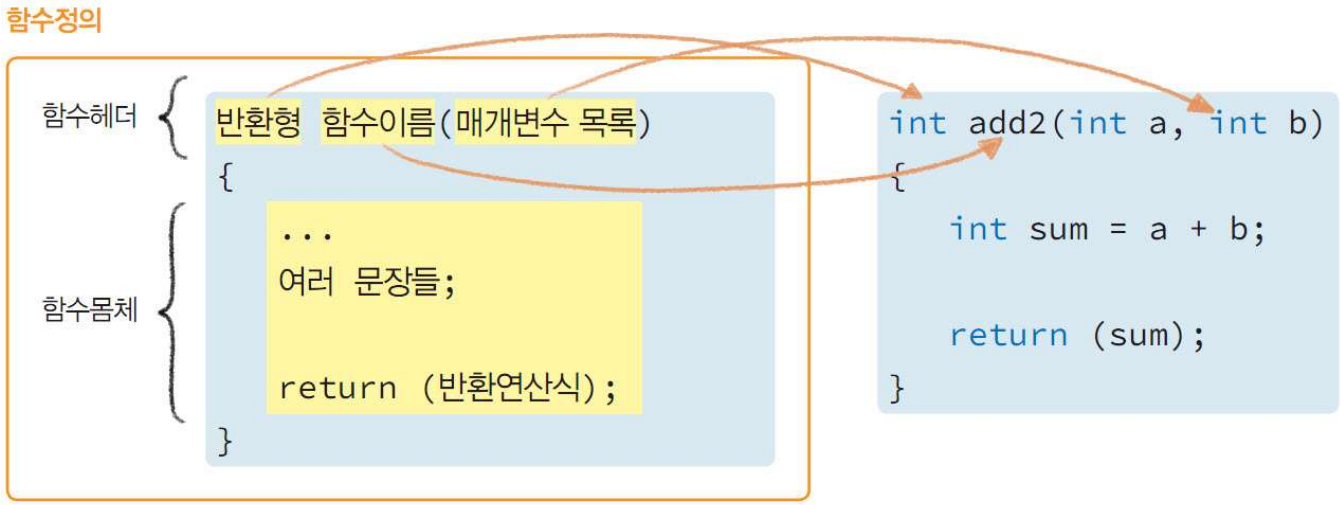


그림 10-4 함수정의 구문

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);
      }
  - 매개변수(parameter variable)
    - (int a, int b)

```
int add2(int a, int b)          함수헤더
{
    int sum = a + b;            함수몸체
    return (sum);
}
```

```
int findMax2(int a, int b)      함수헤더
{
    int max = a > b ? a : b;    함수몸체
    return max;
}
```

그림 10-5 함수정의 예

```
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("%n", min);
    return;          //생략가능
}
```

그림 10-6 반환형과 return 문장

- **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

함수실행

- **정의된 함수를 실행**
  - 프로그램 실행 중에 함수호출(function call)이 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

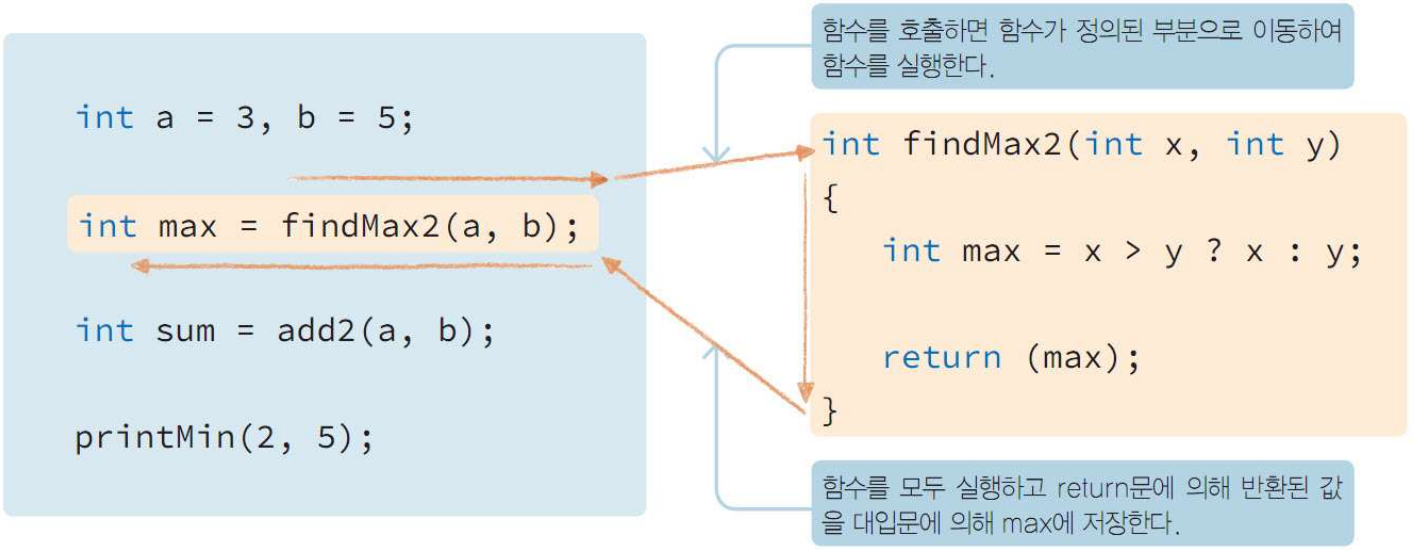


그림 10-8 함수의 호출

함수원형

- **함수원형(function prototype)은 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
  - 변수선언과 같이 함수를 호출하기 전에 반드시 선언
    - 함수의 사용을 미리 컴파일러에게 알리며 또한 프로그램을 이해하기 쉽게 도와주는 역할

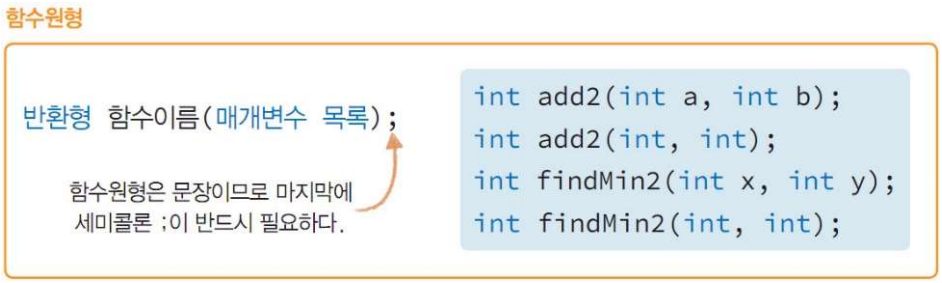


그림 10-9 함수원형 구문

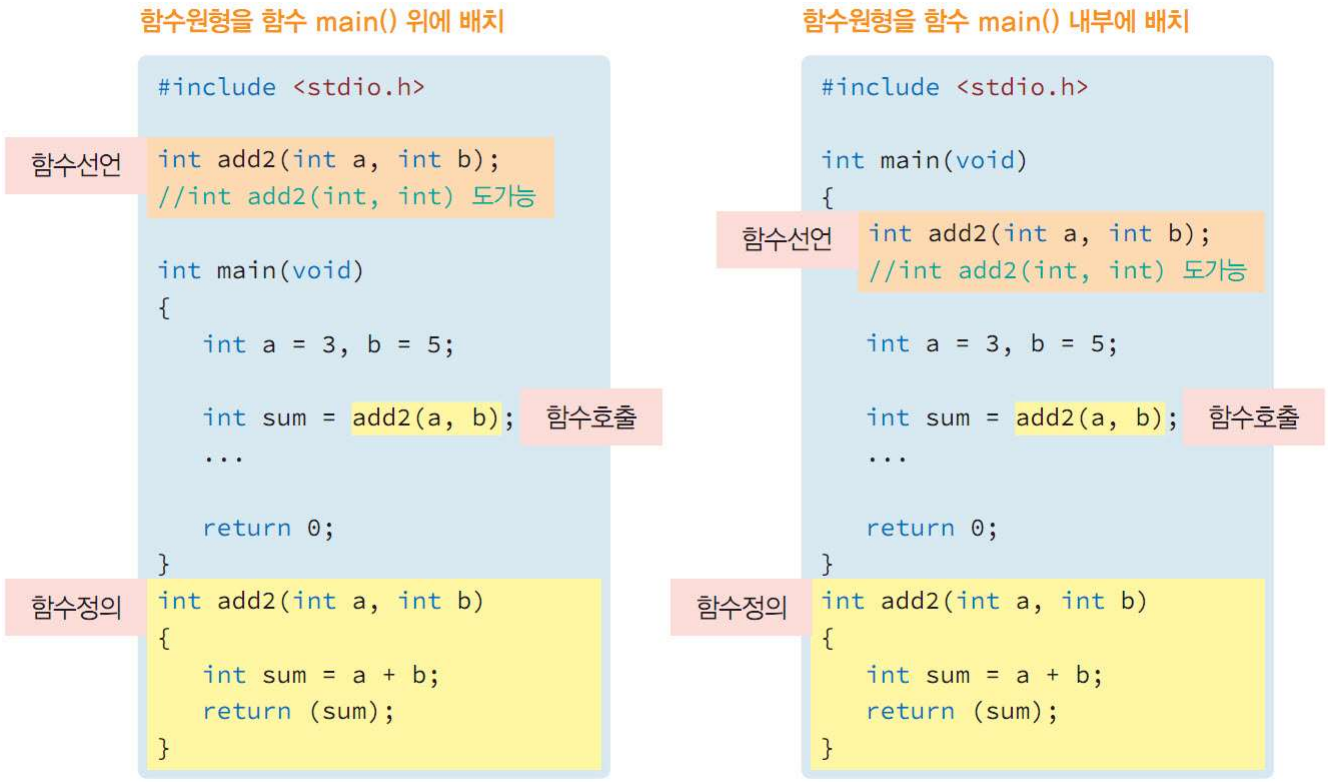


그림 10-10 함수원형 구문과 위치

함수 구현

예제 functionadd2.c

- 함수 add2()와 findMin2()를 구현한 예제

함수호출

- 함수 findMin2()는 함수정의는 구현
  - 함수호출이 없어 실행되지 않음
  - 만일 함수 findMin2()를 실행하려면 main() 함수 내부에서 findMin2()를 호출

실습예제 10-1 functionadd2.c

함수의 정의와 호출

```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)도 가능

   //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);
}
```

설명

```
09    함수선언 문장으로 함수원형을 기술하고 ; 삽입
12    함수 add2()를 호출하여 a+b의 결과를 변수 sum에 저장하는 문장
13    sum 출력
19    함수 add2의 함수헤더
20~24 함수 add2의 함수정의
21    변수 m을 선언하면서 매개변수 a와 b의 합을 저장
23    변수 sum의 값을 반환하는 return 문장으로 괄호는 생략가능하며, 함수호출한 곳으로 매개변수는
      a+b의 결과를 반환
27    함수 findMax2의 함수헤더
28~32 함수 findMax2의 함수정의이나, 위 main() 함수에서 호출이 없으므로 이 함수구현은 실행되지 못함
29    매개변수인 x와 y에서 작은 값을 반환하는 조건 연산자를 이용해 그 결과를 선언되는 변수 min에 저장
31    변수 min의 값을 반환하는 return 문장으로 괄호는 생략가능하며, 함수호출한 곳으로 매개변수는
      x와 y 중에서 작은 값을 반환
```

실행결과

```
합: 8
```

LAB 1에서부터 표준입력으로 받은 양의 정수까지 합을 구하는 함수 getsum()

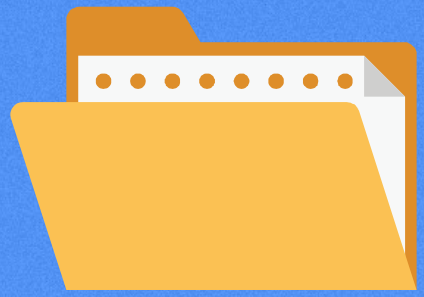
- **함수 getsum(int n)**
  - 1에서 매개변수인 n까지의 합을 구하는 함수
    - 표준입력으로 받은 양의 정수를 구현한 getsum()을 사용하여 적절히 호출하여 그 결과값을 출력
    - 함수 getsum()을 구현
    - 변수 max를 선언하여 표준입력으로 양의 정수를 입력
    - 함수 getsum()을 호출하여 1부터 max까지의 합을 출력
- **결과**
  - 1에서 n까지의 합을 구할 n을 입력하시오. >> 100
  - 1에서 100까지의 합: 5050

Lab 10-1 getsum.c

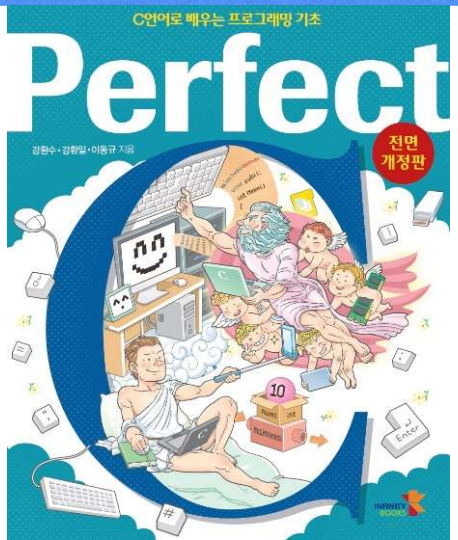
```
01  // file: getsum.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  ________________ //함수원형
06
07  int main(void)
08  {
09     int max = 0;
10
11     printf("1에서 n까지의 합을 구할 n을 입력하시오. >> ");
12     scanf("%d", &max);
13
14     printf("1에서 %d까지의 합: %d\n", _________________; //함수호출
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  }
```

정답

```
05  int getsum(int); //함수원형
14     printf("1에서 %d까지의 합: %d\n", max, getsum(max)); //함수호출
23        sum += i;
```



02. 함수의 매개변수 활용



매개변수와 인자

- **함수 매개변수(parameter)**
  - 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

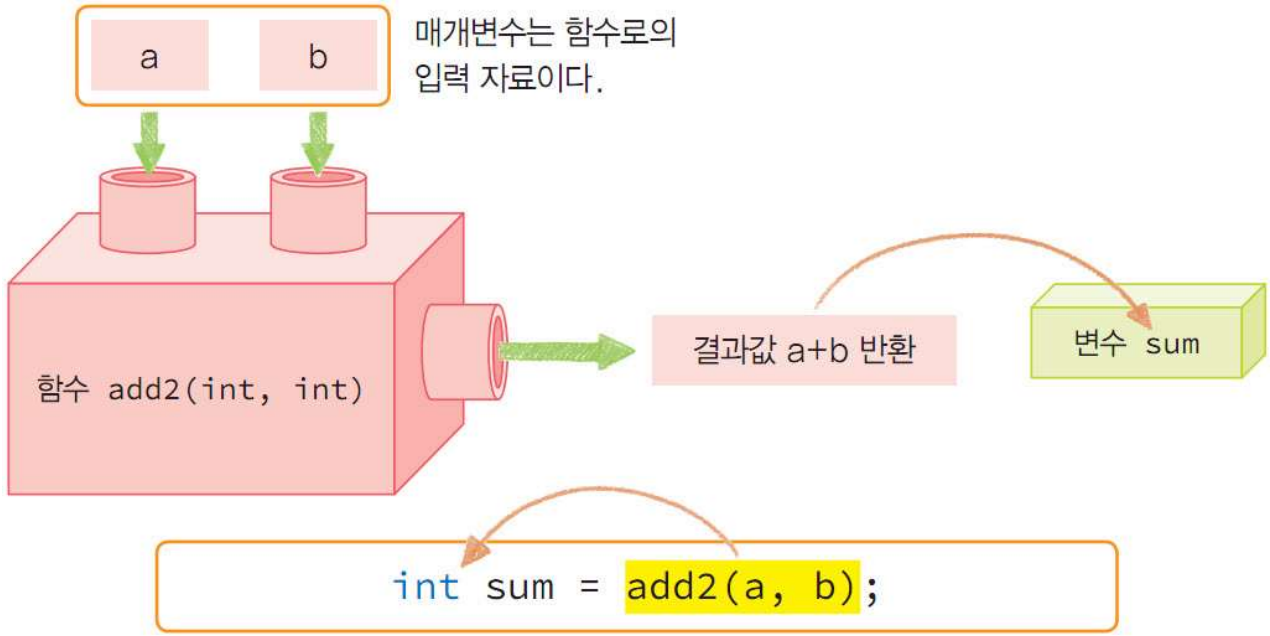
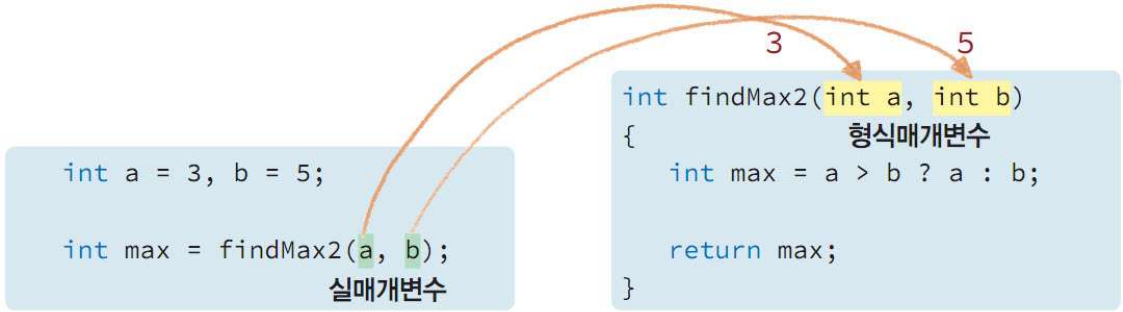


그림 10-11 매개변수와 함수호출

형식매개변수와 실매개변수

- **형식매개변수(formal parameters)**
  - 함수정의에서 기술되는 매개변수 목록의 변수
- **실매개변수(real parameters)**
  - 함수를 호출할 때 기술되는 변수 또는 값
    - 실인자(real argument) 또는 인자(argument)라고도 부름
  - 실인자를 기술할 때는 함수헤더에 정의된 자료유형과 순서가 일치
    - 실인자의 수와 자료형이 다르면 문법오류가 발생
  - 형식인자의 변수는 그 함수가 호출되는 경우
    - 메모리에 할당(allocation)되고 함수가 종료, 메모리에서 자동으로 제거(deallocation)
    - 형식매개변수는 함수 내부 에서만 사용할 수 있는 변수
- **함수호출 findMax2(a, b)**
  - 실인자인 a, b
    - 형식인자 a, b와는 다른 변수
- **값에 의한 호출(call by value)**
  - 실인자의 값이 형식인자의 변수에 각각 복사된 후 함수가 실행



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
```

그림 10-12 형식매개변수와 실매개변수

함수 구현과 호출

예제 functioncall.c

- 매개변수의 사용

실습예제 10-2 functioncall.c

함수의 정의와 호출

```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));

   //반환값이 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
}
```

23 함수 add2의 함수헤더로서, 형식매개변수 a, b
24 매개변수 a와 b의 합을 선언되는 변수 sum에 저장
27 변수 sum의 값을 반환하는 return 문장으로 괄호는 생략가능하며, 함수호출한 곳으로 매개변수는 a+b의 결과를 반환
37~42 함수 findMin2의 함수헤더와 함수정의이나, 위 main() 함수에서 호출이 없으므로 이 함수구현은 실행되지 못함
39 매개변수인 x와 y에서 작은 값을 반환하는 조건 연산자를 이용해 그 결과를 선언되는 변수 min에 저장
41 변수 min의 값을 반환하는 return 문장으로 괄호는 생략가능하며, 함수호출한 곳으로 매개변수는 x와 y 중에서 작은 값을 반환
44 함수 printMIn의 함수헤더로서, 반환값은 없으므로 void, 형식매개변수 a, b
45~50 함수 printMIn의 함수정의
46 변수 min을 선언하면서 조건연산자를 사용하여 매개변수 a와 b 중에서 작은 값을 min에 저장
47 함수구현에서 변수 min 값을 출력
49 함수를 종료하는 return 구문으로 생략 가능

04 함수 add2의 함수선언 문장으로 함수원형을 기술하고 ; 삽입
05 함수 findMax2의 함수선언 문장으로 함수원형을 기술하고 ; 삽입
06 함수 printMIn의 함수선언 문장으로 함수원형을 기술하고 ; 삽입
12 함수 findMax2()를 호출하여 a와 b 중에서 큰 수를 선언되는 변수 max에 저장하는 문장
13 max 출력
14 함수 add2()를 호출하여 a와 b의 합을 반환받아 이 값을 출력문에 사용하여 바로 출력
17 함수 printMIn()를 호출하여 2와 5 중에서 작은 값을 함수정의에서 바로 출력하는데, printMIn()은 반환값이 없으므로 대입문의 오른쪽에 r-value로 사용할 수 없음

실행결과

```
최대: 5
합: 8
최소: 2
```

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

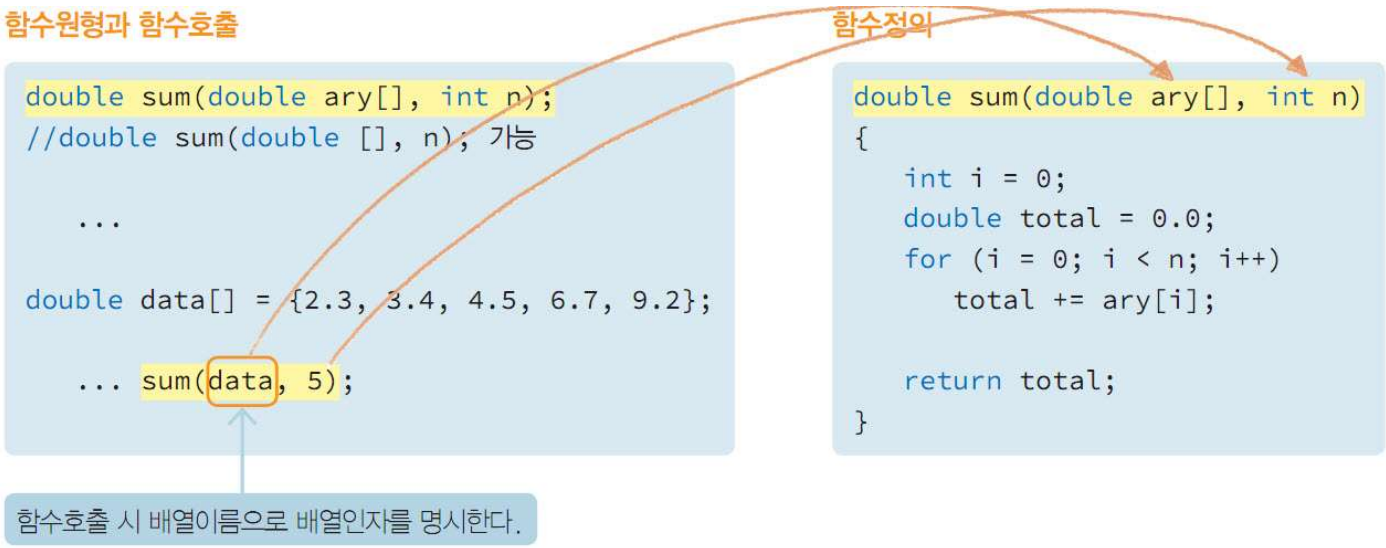


그림 10-13 함수에서 배열 전달을 위한 함수원형과 함수호출 그리고 함수정의

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 상수
    - sum += *(point++)는 불가능
      - 증가연산식 point++의 피연산자로 상수인 point를 사용할 수 없기 때문

```
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++);
```

그림 10-14 간접연산자 *를 사용한 배열원소의 참조방법

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 이용이 가능
    - 연산식 *ary++에서 후위 증가연산자 (ary++)의 우선순위가 가장 높기 때문
      - *(ary++)와 같은 의미

같은 의미로 모두 사용할 수 있다

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

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

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

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

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

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

그림 10-15 함수헤더의 배열 인자와 함수정의에서 다양한 배열원소의 참조방법

배열인자

예제 arrayparam.c

- 함수에서 배열인자의 사용

실습예제 10-3 arrayparam.c

함수에서 배열인자의 사용

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

   return 0;
}

//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;
}
```

설명

함수 sumary의 함수선언 문장으로 함수원형을 기술하고 ; 삽입, 매개변수에서 배열을 사용하는 경우 int ary[]와 int *ary 모두 가능하며, 함수내부에서 배열의 크기를 알 수 없으므로 배열크기를 매개변수 SIZE에 넘김

int 배열 point 선언하면서 초기화

포인터 address에 배열인 point 저장, point는 배열이름으로 상수이고 address는 변수

함수의 매개변수에 배열을 사용하더라도 배열의 첫 원소 주소만이 전달되므로 배열의 크기를 전달해야 배열에 관한 처리가 가능하므로 배열 크기를 구하는 문장으로, 변수 aryLangth에는 배열크기가 저장

합이 저장될 변수 sum 선언

반복 for 문에서 제어변수 i는 배열의 첨자로 사용

배열 원소 point[i]는 *(point+i), *(address+i), *(address++)로 사용 가능하나, point는 상수로 변수가 아니므로 *(point++)는 사용이 불가능

배열 원소의 합이 저장된 sum 출력

포인터 address의 값을 다시 point로 지정

함수 sumary()는 실매개변수는 (point, aryLength)이며, 배열 point의 배열크기의 모든 원소의 합을 반환하므로 point 배열의 합을 출력

실매개변수 point는 &point[0]로도 가능

실매개변수 point는 address로도 가능

함수 sumary()의 함수헤더는 반환값 유형 int, 매개변수는 배열과 배열크기로 각각 int *ary와 int SIZE로 기술하는데, 배열이름인 첫 매개변수는 int ary[]로도 가능

매개변수에서는 배열 int ary[]에서 ary도 변수이므로, 배열 원소 ary[i]는 *(ary+i), *ary++, *(ary++)로 사용 가능

변수 sum의 값을 반환하는 return 문장으로, 함수호출한 곳으로 매개변수는 배열 ary의 모든 원소의 합을 반환

실행결과

```
메인에서 구한 합은 755
함수 sumary() 호출로 구한 합은 755
함수 sumary() 호출로 구한 합은 755
함수 sumary() 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**

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;
}
```

그림 10-16 배열이 함수 인자인 함수원형과 함수호출 그리고 함수정의

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

```
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])
```

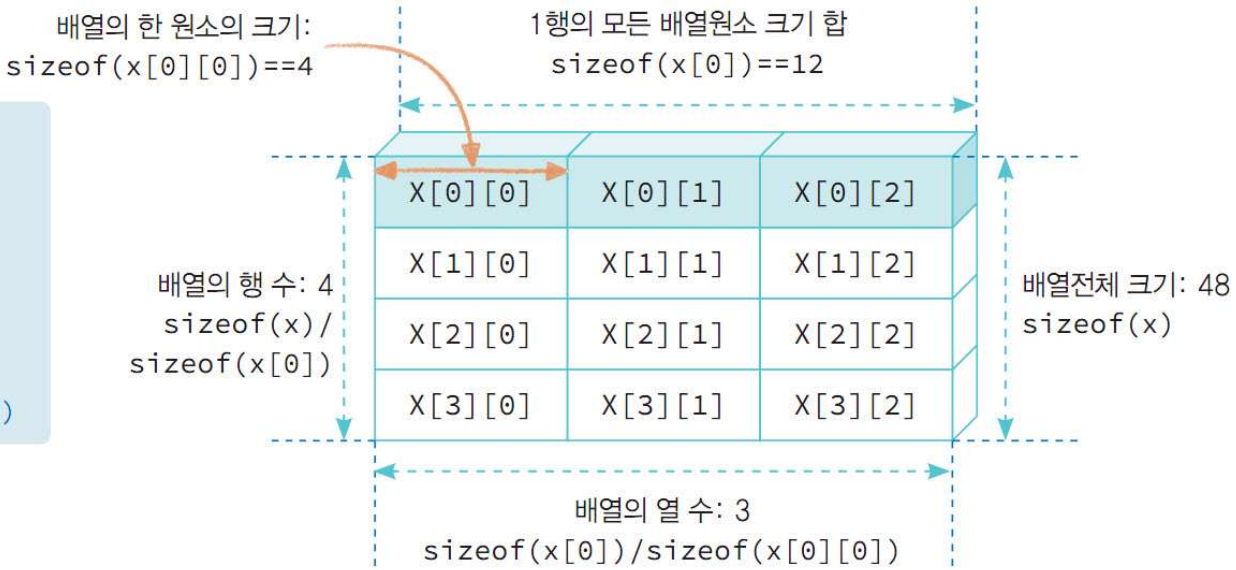


그림 10-17 이차원 배열에서의 행과 열의 수와 전체 배열원소 수 계산

배열인자

예제 twoarrayfunction.c

- 이차원 배열의 활용

```
41      {
42          printf("% d행원소: ", i + 1);
43          for (int j = 0; j < colsize; j++)
44              printf("x[%d][%d] = %5.2lf   ", i, j, data[i][j]);
45          printf("\n");
46      }
47      printf("\n");
48  }
```

설명

```
05  2차원 배열값을 모두 더하는 함수원형 sum, 3열의 이차원 배열을 나타내는 double data[][3]은
    double (*data)[3]로도 가능
07  2차원 배열값을 모두 출력하는 함수원형 printarray, double data[][3]는
    double (*data)[3]로도 가능
12  4 x 3 행렬을 위한 이차원 배열 선언 및 초기화
14  이차원 배열의 행 크기 계산
15  이차원 배열의 열 크기 계산
17  함수 printarray의 첫 번째 인자는 열의 크기가 3인 이차원 배열로 배열이름으로 호출 가능
18  함수 sum의 첫 번째 인자는 열의 크기가 3인 이차원 배열로 배열이름으로 호출 가능하며,
    호출결과인 이차원 배열의 모든 원소의 합을 함수 printf()로 출력
24  함수 sum()의 함수헤더는 반환값 유형 double, 매개변수는 이차원 배열과 배열 행 크기와
    열 크기로 각각 double (*data)[3], int rowsize, int colsize로 기술하는데,
    배열이름인 첫 매개변수는 double data[][3]로도 가능
27  배열의 모든 원소가 저장될 변수 total을 선언하고 초기값 0 저장
29  외부 for 문의 제어변수 i로 이차원 배열의 행을 순회
30  내부 for 문의 제어변수 j로 이차원 배열의 열을 순회
31  변수 total에 data의 모든 원소의 합을 저장
37  함수 printarray()의 함수헤더는 반환값이 없으므로 void, 매개변수는 이차원 배열과
    배열 행 크기와 열 크기로 각각 double (*data)[3], int rowsize, int colsize로
    기술하는데, 배열이름인 첫 매개변수는 double data[][3]로도 가능
41~46 외부 for 문의 함수몸체가 세 문장이므로 반드시 중괄호가 필요
42  행을 구분하기 위한 제목 출력
43  내부 for 문의 함수몸체가 printf() 한 문장이므로 중괄호는 옵션
44  배열의 원소이름과 함께 값을 출력, 첨자 i와 j를 사용
45  다음 행으로 이동하기 위한 출력으로 들여쓰기에 유의
```

실행결과

```
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.
```

실습예제 10-4 twodarrayfunction .c

이차원 배열을 인자로 하는 함수의 정의와 호출

```
01  // file: twodarrayfunction.c
02  #include <stdio.h>
03
04  //2차원 배열값을 모두 더하는 함수원형
05  double sum(double data[][3], int, int);
06  //2차원 배열값을 모두 출력하는 함수원형
07  void printarray(double data[][3], int, int);
08
09  int main(void)
10  {
11     //4 x 3 행렬을 위한 이차원 배열 선언 및 초기화
12     double x[][3] = { { 1, 2, 3 }, { 7, 8, 9 }, { 4, 5, 6 }, { 10, 11, 12 } };
13
14     int rowsize = sizeof(x) / sizeof(x[0]);
15     int colsize = sizeof(x[0]) / sizeof(x[0][0]);
16     printf("2차원 배열의 자료값은 다음과 같습니다.\n");
17     printarray(x, rowsize, colsize);
18     printf("2차원 배열 원소합은 %.3lf 입니다.\n", sum(x, rowsize, colsize));
19
20     return 0;
21  }
22
23  //배열값을 모두 더하는 함수
24  double sum(double (*data)[3], int rowsize, int colsize)
25  //double sum(double data[][3], int rowsize, int colsize)
26  {
27     double total = 0;
28
29     for (int i = 0; i < rowsize; i++)
30        for (int j = 0; j < colsize; j++)
31           total += data[i][j];
32
33     return total;
34  }
35
36  //배열값을 모두 출력하는 함수
37  void printarray(double (*data)[3], int rowsize, int colsize)
38  //void printarray(double data[][3], int rowsize, int colsize)
39  {
40     for (int i = 0; i < rowsize; i++)
```

LAB 배열의 모든 원소를 지정한 만큼 증가시키는 함수 incrementary

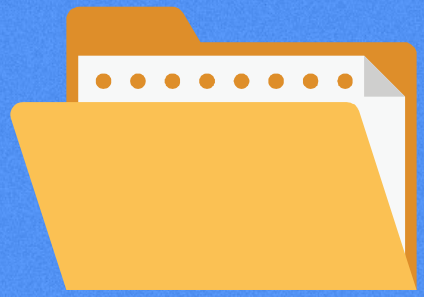
- **함수 incrementary(int ary[], int n, int SIZE)**
  - 배열크기가 SIZE인 배열 ary의 모든 원소를 n만큼 증가시키는 함수
  - 초기화로 선언한 배열 data를 사용하여 모든 원소를 3만큼 증가시키는 함수 호출
    - 변수 aryLength 는 배열의 크기를 저장
    - 함수 prinary(int *data, int SIZE): 배열크기가 SIZE인 배열 ary의 모든 원소를 출력
- **결과**
  - 4 7 2 3 5
  - 배열 원소에 각각 3을 더한 결과:
  - 7 10 5 6 8

Lab 10-2 incrementary.c

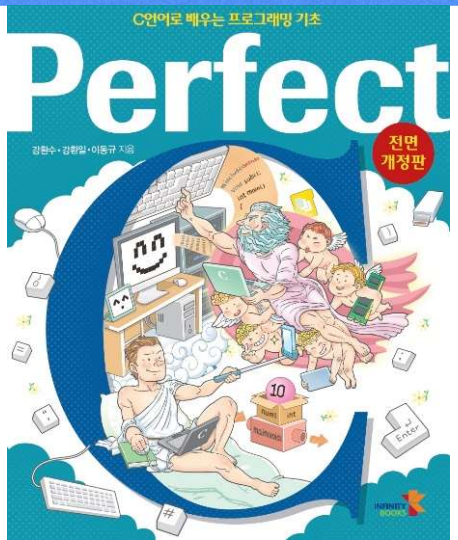
```c
// file: incrementary.c
#include <stdio.h>

//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 3을 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");
}
```

정답

```c
void incrementary(int ary[], int n, int SIZE);
incrementary(data, 3, aryLength);
   *(ary + i) += n; 또는 ary[i] += n;
```



03. 재귀와 라이브러리 함수



재귀 특성

- **재귀 함수(recursive function)**
  - 함수구현에서 자신 함수를 호출하는 함수
  - 재귀적 특성을 표현하는 알고리즘
    - 재귀 함수를 이용하면 문제를 쉽게 해결할 수 있고 이해하기도 쉬움
- **수학 수식에서 재귀적 특성**
  - n 계승(n factorial)을 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

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

그림 10-18 n!의 정의와 재귀 특성

```
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));
}
```

그림 10-19 n!을 위한 함수 factorial()

재귀 함수

예제 factorial.c

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

실습예제 10-5 factorial.c

n! 재귀함수

```
01  // file: factorial.c
02  #include <stdio.h>
03
04  int factorial(int); //함수원형
05
06  int main(void)
07  {
08     for (int i = 1; i <= 10; i++)
09        printf("%2d! = %d\n", i, factorial(i));
10
11     return 0;
12  }
13
14  // n! 구하는 재귀함수
15  int factorial(int number)
16  {
17     if (number <= 1)
18        return 1;   ← 조건식을 만족하면 더 이상 자기 자신인 함수 factorial()을 호출하지 않는다.
19     else
20        return (number * factorial(number - 1));
21  }
```

설명

```
04  n!을 구하기 위한 함수 factorial()의 함수원형
08  반복 for 문에서 제어변수 i를 사용하여 1에서 10까지 반복
09  함수 factorial(i)를 호출하여 그 결과를 출력
15  함수 factorial()의 함수헤더는 반환값인 n!의 유형 int, 매개변수는 n!에서의 n에
    해당하는 number
18  1!은 1이므로 1 반환
20  n! = n * (n-1)!이므로 재귀함수 호출로 number * factorial(number - 1)을 반환
```

실행결과

```
 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

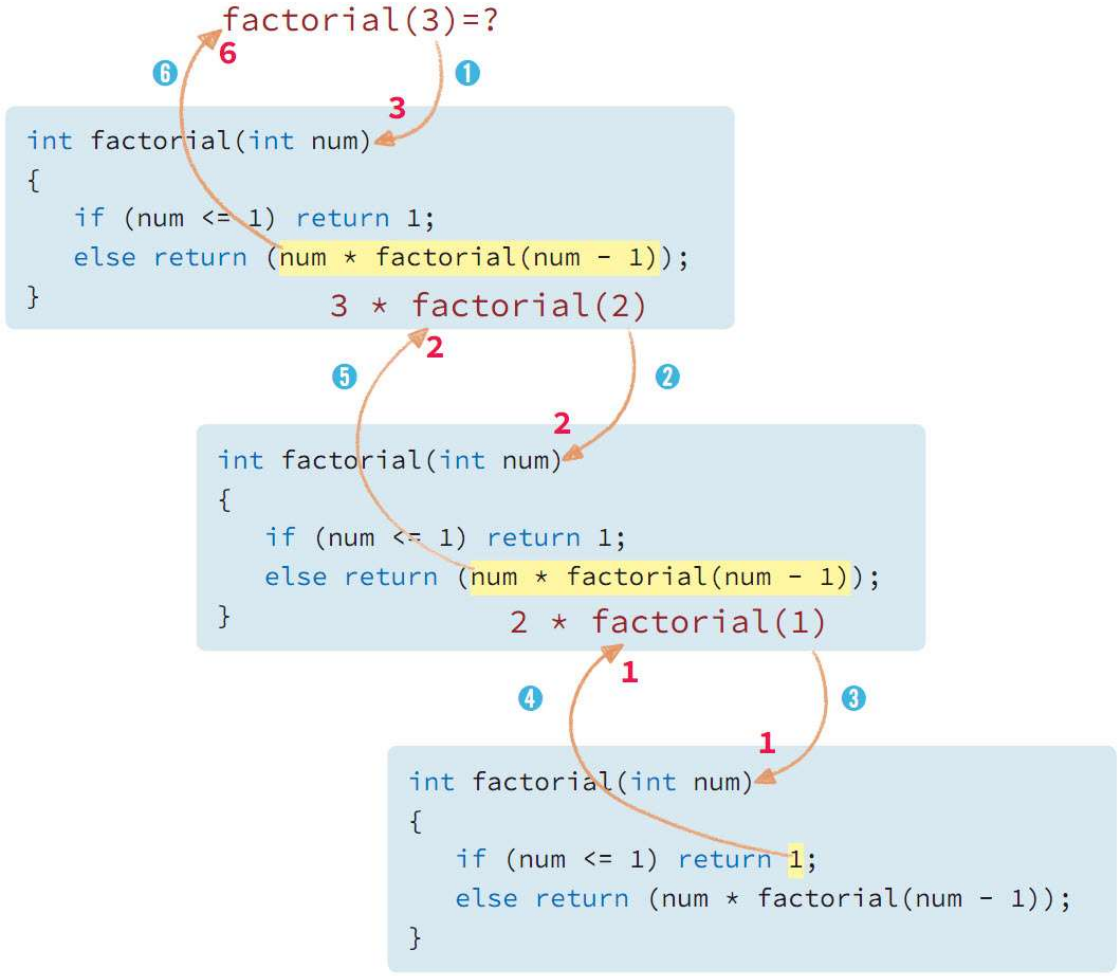


그림 10-20 재귀함수의 실행

난수(random 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

```
#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

그림 10-21 함수 rand()의 활용

난수 출력

예제 rand.c

- 난수 5개 출력

실습예제 10-6 rand.c

난수를 위한 함수 rand()의 이용

```
01  // file: rand.c
02  #include <stdio.h>
03
04  #include <stdlib.h> //rand()위한 헤더파일 포함
05
06  int main(void)
07  {
08     printf("0 ~ %5d 사이의 난수 5개: rand()\n", RAND_MAX);
09     for (int i = 0; i < 5; i++)
10        printf("%5d ", rand());
11     puts("");
12
13     return 0;
14  }
```

설명

04 함수 rand()를 위한 헤더파일 stdlib.h 포함
08 상수 RAND_MAX는 32767
09 반복 for 문에서 제어변수 i를 사용하여 0에서 4까지 5회 반복
10 함수 rand() 호출로 난수 출력

실행결과

```
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) + a 을 이용

```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.

그림 10-22 1에서 100사이의 난수 생성 모듈

매번 다른 난수 출력

예제 srand.c

- 시드 지정 후 난수 출력

실습예제 10-7 srand.c

srand()로 시드값을 먼저 지정한 후 1에서 100사이의 난수를 생성

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

   return 0;
}
```

설명

```
03  함수 srand()와 rand()를 위한 헤더파일 stdlib.h 포함
05  함수 time()을 위한 헤더파일 time.h 포함
07  매크로 상수 MAX를 100으로 정의
11  변수 seconds를 선언하면서 라이브러리 time()을 호출하여 그 결과를 저장
12  변수 seconds를 실매개변수로 함수 srand(seconds)를 호출
14  제목으로 상수 MAX 출력
15  반복 for 문에서 제어변수 i를 사용하여 0에서 4까지 5회 반복
16  함수 rand() 호출로 1에서 MAX 사이의 난수 출력
```

실행결과

```
1 ~   100 사이의 난수 5개:
   88    10    66    90     3
```

수학과 문자 라이브러리 함수

예제 srand.c

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

표 10-1 수학 관련 함수

함수	처리 작
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

실습예제 10-8 math.c

다양한 수학 관련 함수 이용

```
01  // file: math.c
02  #include <stdio.h>
03
04  #include <math.h> //수학 관련 다양한 함수헤더 포함 헤더파일
05
06  int main(void)
07  {
08     printf("  i    i제곱    i세제곱   제곱근(sqrt)\n");
09     printf("-------------------------------------\n");
10     for (int i = 3; i < 7; i++)
11        printf("%3d %7.1f %9.1f %9.1f\n", i, pow(i, 2), pow(i, 3), sqrt(i));
12     printf("\n");
13
14     printf("exp(1.0) == %5.2f, ", exp(1.0));
15     printf("pow(2.72, 1.0) == %5.2f, ", pow(2.72, 1.0));
16     printf("sqrt(49) == %5.2f\n", sqrt(49));
17     printf("abs(-10) == %3d, ", abs(-10));
18     printf("ceil(7.1) == %5.2f, ", ceil(7.1));
19     printf("floor(6.9) == %5.2f\n", floor(6.9));
20
21     return 0;
22  }
```

설명

```
04     함수 exp(), pow(), sqrt() 등 다양한 수학 관련 함수 사용을 헤더파일 math.h 포함
08~09  제목 출력
10     반복 for 문에서 제어변수 i를 사용하여 3에서 6까지 4회 반복
11     함수 pow()와 sqrt() 호출하여 출력
14     함수 exp(x)는 자연수인 e = 2.72에서 e^x을 반환하므로 (2.72)^1
15     함수 pow(2.72, 1.0)은 (2.72)^1을 반환
16     함수 sqrt(49)는 √49 반환
17     함수 abs(-10)는 -10의 절대값을 반환
18     함수 ceil(7.1)은 7.1의 천정값인 8.0을 반환하는데,
       천정값 ceil(x)이란 x보다 작지 않은 가장 작은 정수
18     함수 floor(6.9)는 6.9의 바닥값인 6.0을 반환하는데,
       바닥값 floor(x)이란 x보다 크지 않은 가장 큰 정수
```

실행결과

```
  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
```

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

표 10-2 헤더파일 ctype.h의 주요 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

문자 라이브러리 함수

예제 char.c

- 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

실습예제 10-9 char.c

문자 관련 함수 이용

```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;
}
```

설명

```
05    함수 isalpha(), isupper(), tolower(), toupper() 등 다양한 문자 관련 함수 사용을
      위해 헤더파일 ctype.h 삽입
07    함수 print2char()의 함수선언인 함수원형
11    문자 하나를 입력 받을 변수 ch 선언
13    제목 출력
14~23 do while 문으로 반복몸체는 16행부터 22행까지
16    입력 방식 출력
17    문자 하나를 표준입력으로 변수 ch에 저장
18    문자 하나를 입력한 후 enter키를 반드시 누르도록 하여 이 enter키를 하나 받아들여 버리는 기능
19    입력 문자가 영문자 알파벳인지를 검사
20    입력 문자가 영문자 알파벳이면 대문자는 소문자로, 소문자는 대문자로 변환하여 출력하는
      함수 print2char(ch) 호출
22    입력 문자가 영문자 알파벳이 아니면 바로 문자 그대로 출력
23    조건식 (ch != 'x' && ch != 'X')은 입력인 ch가 x와 X가 아니면 반복을 계속하고,
      x 또는 X이면 종료
28    대문자는 소문자로, 소문자는 대문자로 변환하여 출력하는 함수 print2char(ch)의 함수헤더
30    매개변수 ch가 대문자인지 검사
31    매개변수 ch가 대문자이면 함수 tolower(ch)를 호출하여 소문자로 변환하여 출력
33    매개변수 ch가 소문자이면 함수 toupper(ch)를 호출하여 대문자로 변환하여 출력
```

다양한 헤더파일

- **라이브러리 함수를 제공**
  - 여러 라이브러리 함수를 위한 함수원형과 상수, 그리고 매크로가 여러 헤더파일

표 10-3 여러 라이브러리를 위한 헤더 파일

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 1에서 100사이의 난수 알아 맞추기

- **1에서 100사이의 한 난수를 미리 저장하여, 이 정수를 알아 맞추는 프로그램**
  - 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

Lab 10-3 numberguess.c

```
01  // file: numberguess.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  #include <stdlib.h> //rand(), srand()를 위한 헤더파일 포함
06  #include <time.h>   //time()을 위한 헤더파일 포함
07
08  #define MAX 100
09
10  int main(void)
11  {
12     int guess, input;
13
14     srand((long)time(NULL));
15     guess = ________________________;
16
17     printf("1에서 %d 사이에서 한 정수가 결정되었습니다.\n", MAX);
18     printf("이 정수는 무엇일까요? 입력해 보세요. : ");
19
20     while ( scanf(_____________________________) ) {
21        if (input > guess)
22           printf("입력한 수보다 작습니다. 다시 입력하세요. : ");
23        else if (input < guess)
24           printf("입력한 수보다 큽니다. 다시 입력하세요. : ");
25        else
26        {
27           puts("정답입니다.");
28           _________________;
29        }
30     }
31
32     return 0;
33  }
```

정답

```
15  guess = rand() % MAX + 1;
20  while ( scanf("%d", &input) ) {
28        break;
```

Thank you