C언어로 배우는 프로그래밍 기초

Perfect C 3판

INFINITY BOOKS

제 9 장 함수 기초

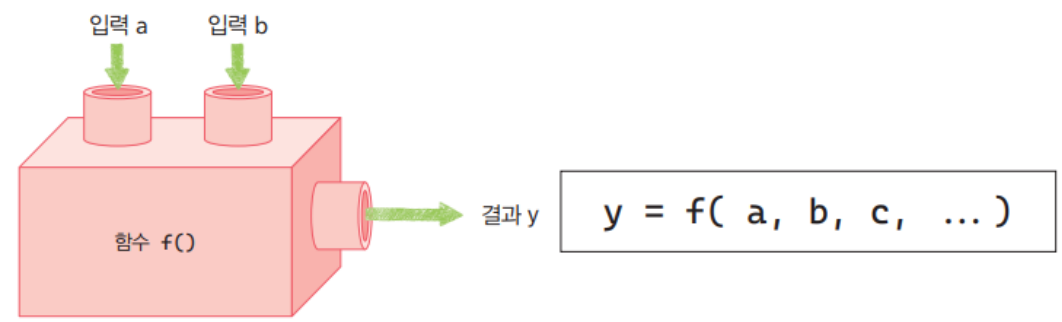
01 함수 정의와 호출
02 함수의 매개변수 활용
03 재귀와 라이브러리 함수

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 함수몸체
  - 함수 호출 방법 및 실행 순서
  - 함수원형 구문과 필요성
- 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 시드 값을 위한 srand()의 사용
  - 문자와 수학 관련 함수의 이용

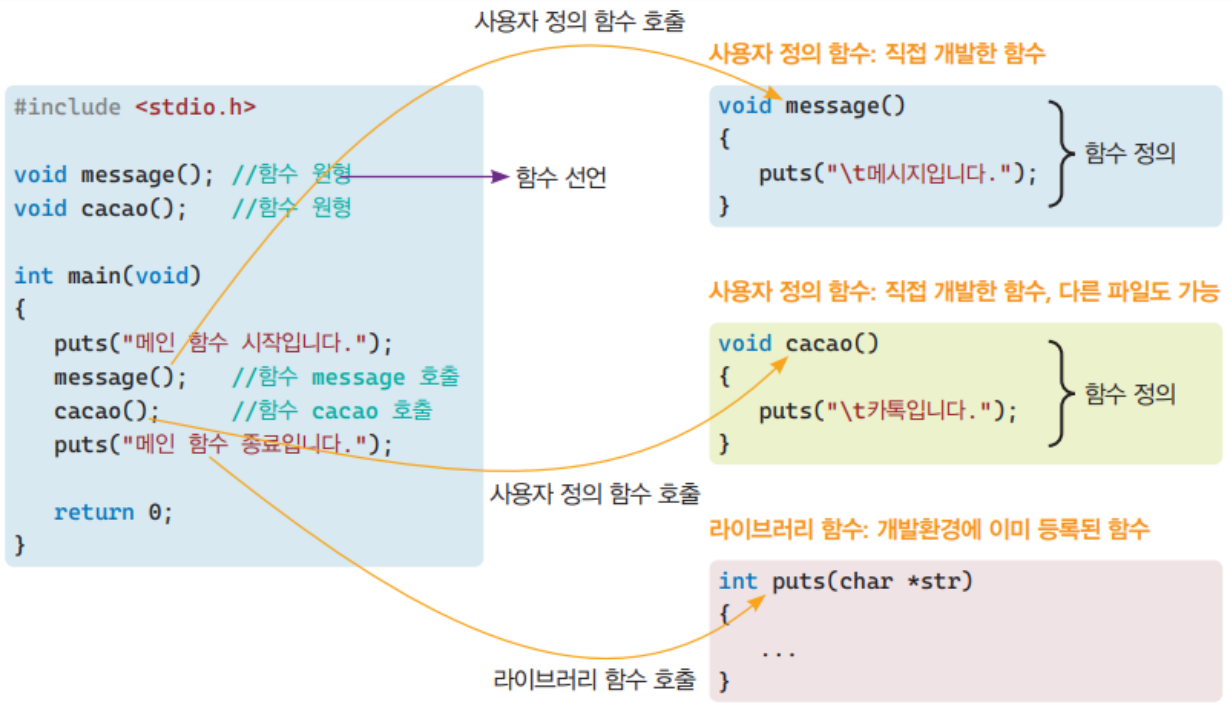
함수의 이해

- **함수 개념**
  - 함수(function)
    - **프로그램에서 원하는 특정한 작업을 수행하도록 설계된 독립된 프로그램 단위**
      - **필요한 입력을 받아 원하는 어떤 기능을 수행한 후 결과를 반환(return)하는 프로 그램 단위**
  - 라이브러리 함수(library function)와 사용자 정의 함수(user defined function)로 구분
    - **사용자 정의 함수**
      - **필요에 의해서 개발자가 직접 개발하는 함수**
- **C 프로그램**
  - 여러 함수로 구성되는 프로그램
    - **최소한 main() 함수와 필요한 다른 함수로 구성되는 프로그램**
  - 함수 main()
    - **이름이 지정된 함수로서 프로그램의 실행이 시작되는 특수한 함수**



함수 정의와 호출

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

- **함수 message()**
  - 함수 main()이 구현된 동일한 파일 01basefunc.c에서 구현
  - 함수 cacao()는 다른 파일인 01catalk.c에서 구현

실습예제 9-1 | Prj01 | 01basefunc.c, 01catalk.c | 함수 정의와 호출 | 난이도: ★

소스파일 01basefunc.c

```c
#include <stdio.h>

void message();       //함수원형
void cacao();         //함수원형

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.");
}
```

함수원형은 함수 호출이 되는 파일의 상단에 배치.

함수 message()의 구현도 다른 01catalk.c나 다른 파일에서 구현 가능

소스파일 01catalk.c

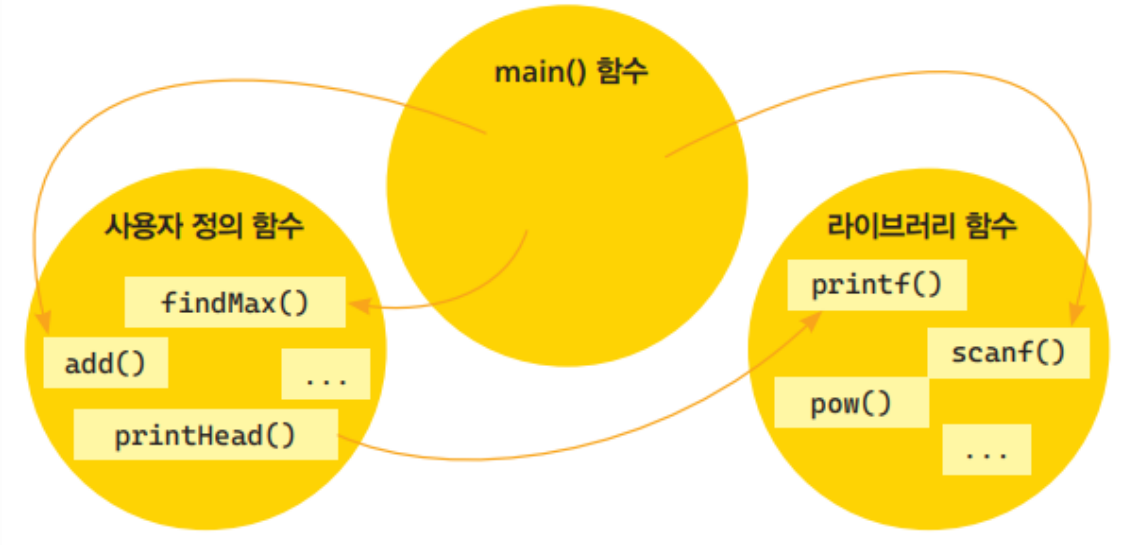
```c
// 함수 main()과 다른 파일에 함수 구현 가능
void cacao()
{
   puts("\t카톡입니다.");
}
```

결과

```
메인 함수 시작입니다.
        메시지입니다.
        카톡입니다.
메인 함수 종료입니다.
```

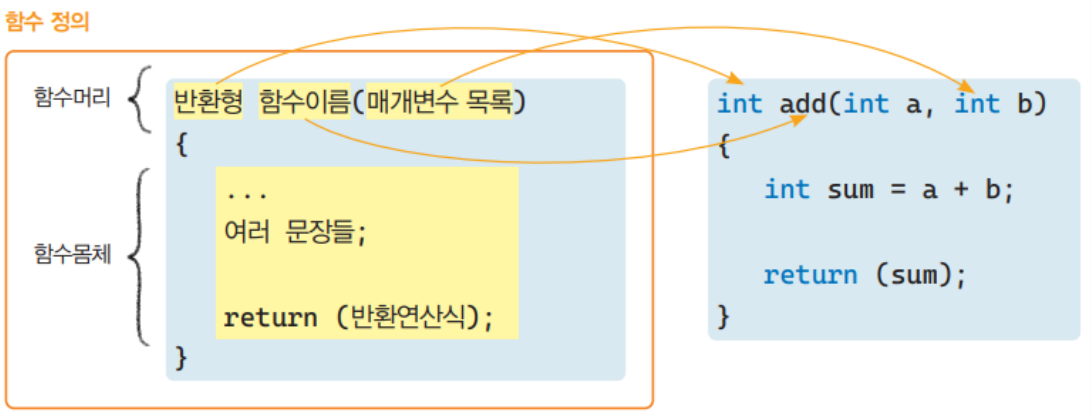
절차적 프로그래밍

- **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 프로그래밍3(procedural programming) 방식**
  - 함수 중심의 프로그래밍 방식
  - 모듈화 프로그램 (modular program) 또는 구조화된 프로그램(structured program)
    - **적절한 함수로 잘 구성된 프로그램**



함수 정의 구문

- **함수머리(function header)와 함수몸체(function body)로 구성**
- **함수머리(function header)**
  - 반환형과 함수이름, 매개변수 목록으로 구성
  - 반환형: 함수 결과값의 자료형
    - **간단히 반환형**
  - 함수이름: 식별자의 생성규칙
  - 매개변수 목록: '자료형 변수이름'의 쌍
    - **필요한 수만큼 콤마로 구분하여 기술**
- **함수몸체(function body)**
  - {...}와 같이 중괄호로 시작하여 중괄호로 종료
  - 함수가 수행해야 할 문장들로 구성
  - 마지막은 대부분 결과값을 반환하는 return 문장으로 종료
  - 결과값이 없다면 생략 가능



반환형과 return

- **함수에서 반환값을 전달하는 목적과 함께 함수의 작업 종료를 알리는 문장**
  - 함수가 반환값이 없다면 반환형으로 void를 기술

```
int findMin(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;       //생략 가능
}
```

return 문장

return (반환연산식);

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

함수원형

- **function prototype**
  - 함수도 정의된 함수를 호출하기 이전에 필요
  - 함수원형은 함수를 선언하는 문장
- **구문**
  - 함수머리에 세미콜론을 넣은 문장
    - **int add(int a, int b);**
    - **int add(int, int);**

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);
```

함수원형을 함수 main() 위에 배치

```
#include <stdio.h>

int add(int a, int b);          // 함수 선언
//int add(int, int)도 가능

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

   int sum = add(a, b);         // 함수 호출
   ...

   return 0;
}
int add(int a, int b)           // 함수 정의
{
   int sum = a + b;
   return (sum);
}
```

함수원형을 함수 main() 내부에 배치

```
#include <stdio.h>

int main(void)
{
   int add(int a, int b);       // 함수 선언
   //int add(int, int)도 가능

   int a = 3, b = 5;

   int sum = add(a, b);         // 함수 호출
   ...

   return 0;
}
int add(int a, int b)           // 함수 정의
{
   int sum = a + b;
   return (sum);
}
```

함수의 정의와 호출

실습예제 9-2 | Prj02 | 02funcadd.c | 함수의 정의와 호출 | 난이도: ★

```c
#include <stdio.h>

//int add(int a, int b);   //이 위치도 가능

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

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

   return 0;
}

//함수 add의 함수 구현 또는 함수 정의 부분
int add(int a, int b)
{
   int sum = a + b;

   return (sum); //괄호는 생략 가능
}

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

   return (min);
}
```

변수 sum의 값을 반환하는 return 문장으로 괄호는 생략 가능하며, 함수 호출한 곳으로 매개변수는 a+b의 결과를 반환

결과

```
합: 8
```

LAB 함수 int pow()

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

Lab 9-1 lab1intpower.c 난이도: ★★

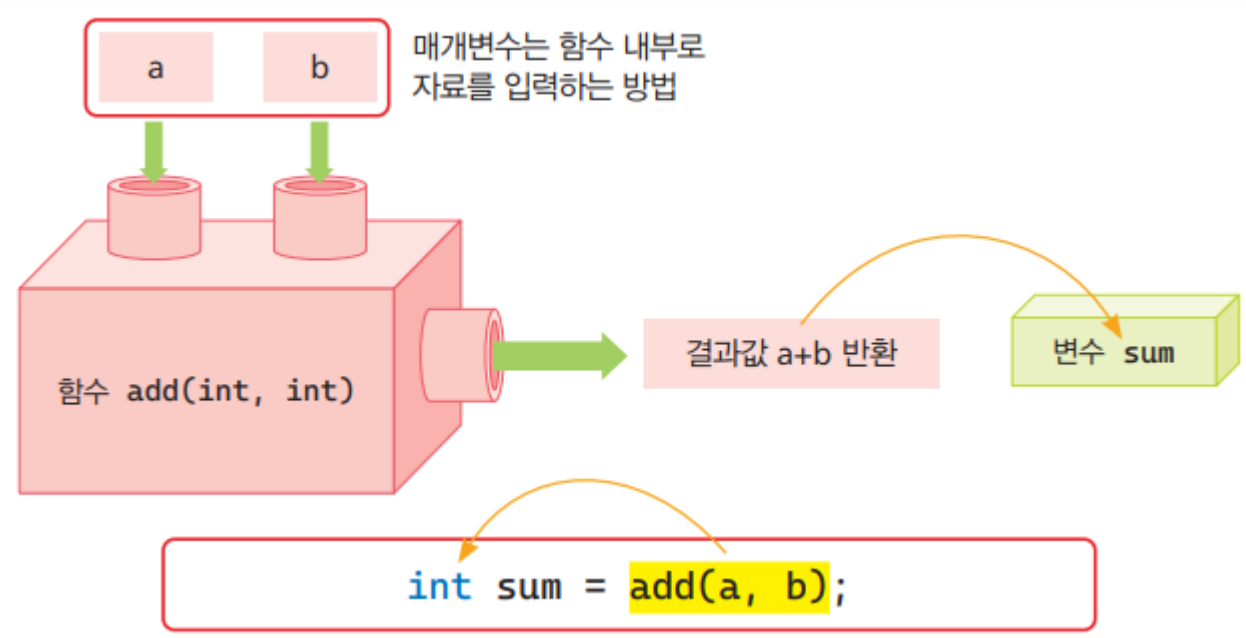
```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  [                    ];
05
06  int main(void)
07  {
08     int m, n;
09
10     printf("정수 m을 n번 제곱합니다.\n");
11     printf("정수 m n을 계속 입력 >>> ");
12     scanf("%d %d", &m, &n);
13
14     //함수 호출
15     printf("\n%d의 %d 제곱은 %d입니다.\n", m, n, [                    ]);
16
17     return 0;
18  }
19
20  int intpow(int m, int n)
21  {
22     int mult = 1, i = 1;
23
24     for (i = 1; i <= n; i++) {
25        [          ];
26     }
27
28     return mult;
29  }
```

정답

```
04  int intpow(int m, int n);
15     printf("\n%d의 %d 제곱은 %d입니다.\n", m, n, intpow(m, n));
25        mult *= m;
```

매개변수 정의

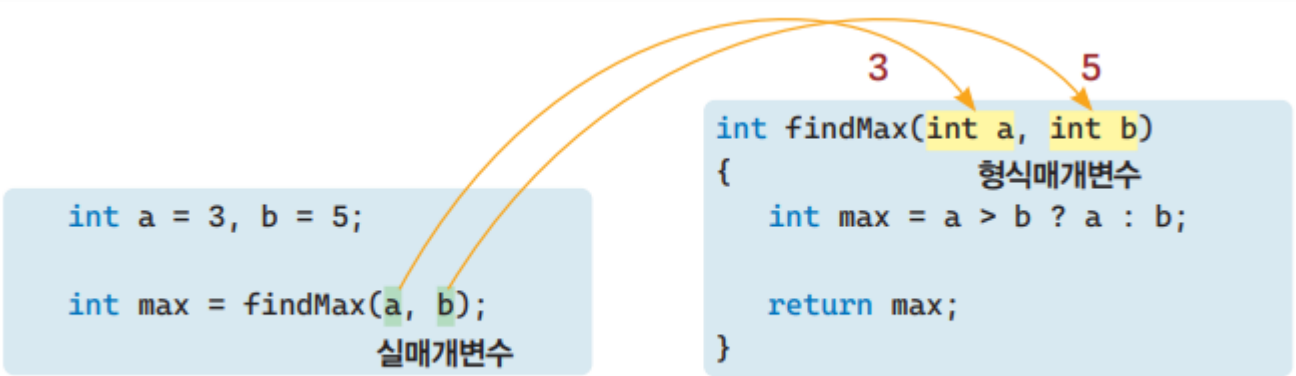
- **함수 매개변수(parameter)**
  - 함수를 호출하는 부분에서 함수몸체로 값을 전달할 목적으로 이용
  - 자료형과 변수명의 목록으로 표시
  - 필요 없으면 키워드 void를 기술
- **반환값**
  - 함수를 호출하는 부분에서 함수가 작업을 수 행한 후, 다시 함수를 호출한 영역으로 보내는 결과값



형식매개변수와 실매개변수

- **형식매개변수(formal parameters)**
  - 함수 정의에서 기술되는 매개변수 목록의 변수
  - 함수 내부에서만 사용될 수 있는 변수
- **실매개변수(real parameters)와 실인자(real argument)**
  - 함수를 호출할 때 기술되는 변수 또는 값
  - 간단히 인자(argument)라고도 부름

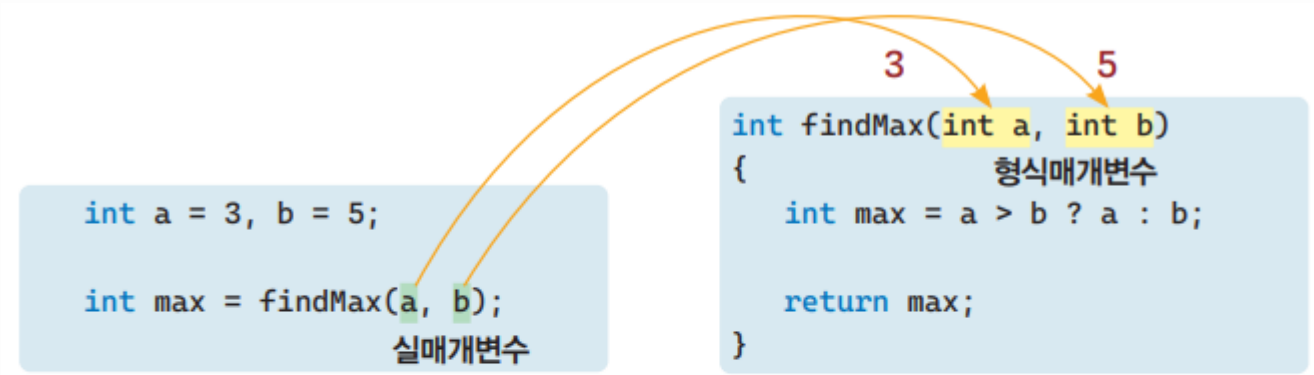
- **일반적으로 매개변수, 인자, 인수**
  - 구분하지 않고 사용하는 경우도 많음



함수 호출 시 매개변수의 수와 자료형이 다르면 문법오류가 발생한다.

값에 의한 호출(call by value)

- **함수가 호출되는 경우 실인자의 값이 형식인자의 변수에 각각 복사된 후 함수가 실행**
  - 형식인자 a, b와는 전혀 다른 변수
  - 함수에서 매개변수(일반 변수)의 값을 수정하더라도 함수를 호출한 곳에서의 실제 변수의 값은 변화되지 않는 특징



함수 호출 시 매개변수의 수와 자료형이 다르면 문법오류가 발생한다.

```
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
```

함수의 정의와 호출

실습예제 9-3 | Prj03 | 03functioncall.c, 03others.c | 함수의 정의와 호출 | 난이도: ★

소스파일 03functioncall.c

```c
#include <stdio.h>

int add(int a, int b);          //int add(int, int)도 가능
int findMax(int, int);          //int findMax(int a, int b)도 가능
void printMin(int, int);        //int printMin(int a, int b)도 가능

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));

   //반환값이 없는 함수 호출은 일반문장처럼 사용
   printMin(a, b);

   return 0;
}

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

함수 printMin()을 호출하여 a와 b 중에서 작은 값을 함수 정의에서 바로 출력하는데, printMin()은 반환값이 없으므로 대입문의 오른쪽에 r-value로 사용할 수 없음

소스파일 03others.c

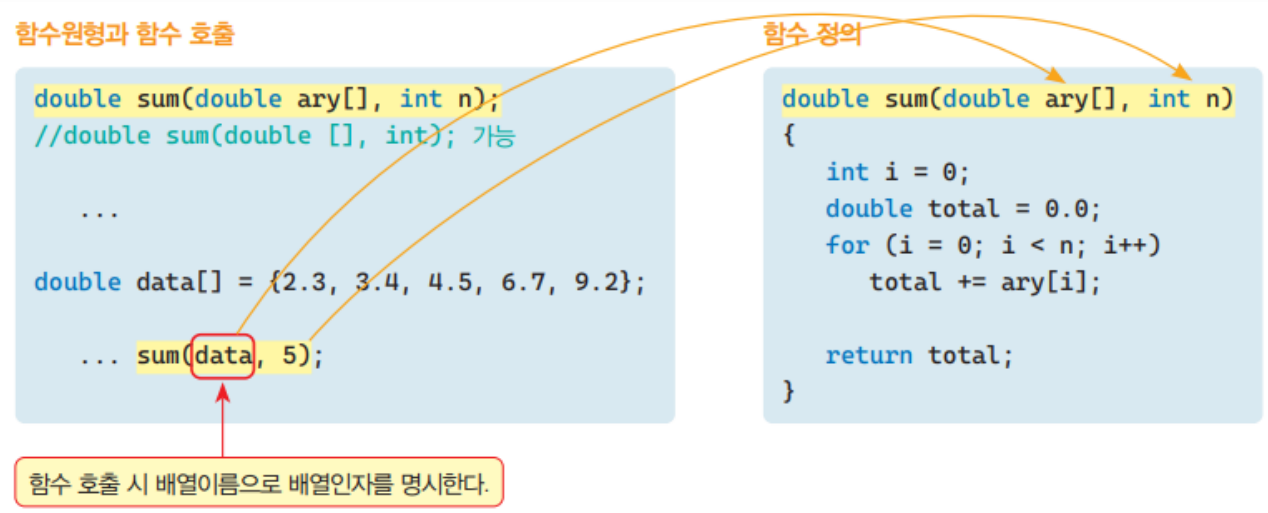
```c
//함수 add, findMax, findMin, printMin 구현
int add(int a, int b)
{
   int sum = a + b;

   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);
}
```

결과

```
최대: 15
합: 25
최소: 10
```

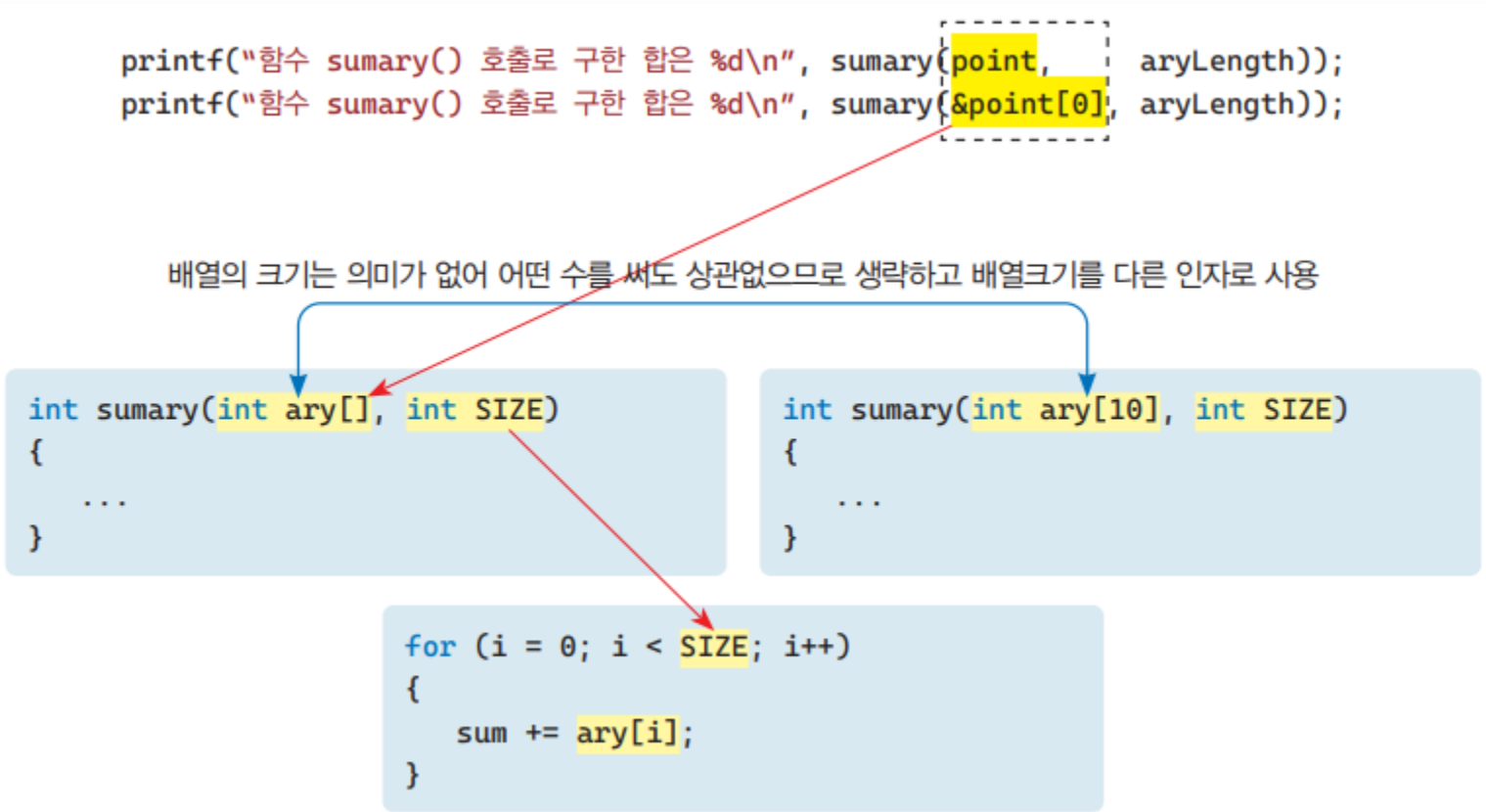
매개변수로 배열 사용

- **함수의 매개변수로 배열을 전달**
  - 한 번에 여러 개의 변수를 전달하는 효과
- **함수 sum()**
  - 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수
  - 형식매개변수는 실수형 배열 double ary[]와 배열크기 int n
    - **첫 번째 형식매개변수에서 배열 자체에 배열크기를 기술하는 것은 아무 의미가 없음**
    - **double ary[5]보다는 double ary[]라고 기술하는 것을 권장**
  - 실제로 함수 내부에서 실인자로 전달되는 배열크기를 알 수 없으므로
    - **배열크기를 두 번째 인자로 사용**



배열을 매개변수로 사용하는 방법

- **함수크기를 제외한 int ary[]로 기술**
  - 배열의 크기도 매개변수로 함께 전달
  - 함수 이름인 point 또는 &point[0]를 사용



함수에서 배열인자를 사용

실습예제 9-4 | Prj04 | 04aryparam.c | 함수에서 배열인자를 사용 | 난이도: ★

```c
#include <stdio.h>

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

   return 0;
}

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

   return sum;
}
```

배열의 실매개변수 사용은 point와 &point[0]로도 가능

함수의 매개변수에 배열을 사용하더라도 배열의 첫 원소 주소만이 전달되므로 배열의 크기를 전달해야 배열에 관한 처리가 가능함. 이것은 배열 크기를 구하는 문장으로, 변수 aryLength에는 배열크기가 저장됨

이 곳에서는 함수를 호출한 곳에서의 배열 전체 크기를 알 수 없음

결과

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

이차원 배열을 함수 인자로 이용하는 방법

- **함수원형과 함수정의의 헤더**
  - 첫 번째 대괄호 내부의 크기를 제외한 다른 모든 크기는 반드시 기술

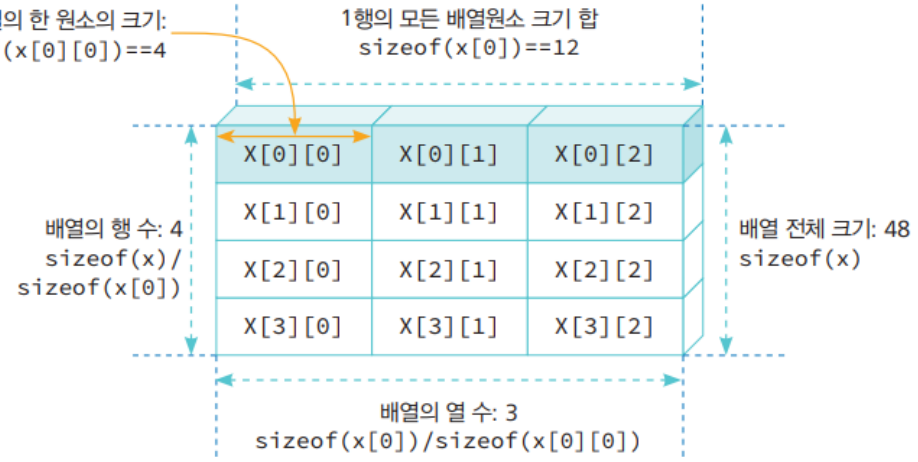
함수원형과 함수 호출

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

...
double x[][3] = { {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);

... sum(x, rowsize, colsize) ...

...
```

함수 정의

```
//2차원 배열값을 모두 출력하는 함수
void printarray(double data[][3], int rowsize, int colsize)
{

...

}

//2차원 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;
}
```

```
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])
```



2차원 배열을 인자로 하는 함수의 정의와 호출

실습예제 9-5 | Prj05 | 05twodaryfunc.c | 2차원 배열을 인자로 하는 함수의 정의와 호출 | 난이도: ★★

```c
#include <stdio.h>

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

int main(void)
{
   //3 x 2 행렬을 위한 2차원 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;
}

//배열값을 모두 더하는 함수
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;
}

//배열값을 모두 출력하는 함수
void printarray(double data[][2], 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");
}
```

열의 크기는 명시되어야 함

결과

```
2차원 배열의 자료값은 다음과 같습니다.
1행원소: x[0][0] = 11.00   x[0][1] = 12.00
2행원소: x[1][0] = 21.00   x[1][1] = 22.00
3행원소: x[2][0] = 31.00   x[2][1] = 32.00
2차원 배열 원소합은 129.000 입니다.
```

LAB 배열의 모든 원소를 지정한 수만큼 증가시키는 함수 increment()

- **함수 increment(int ary[], int n, int SIZE)**
  - 배열크기가 SIZE인 배열 ary의 모든 원소를 n만큼 증가시키는 함수
- **초기화로 선언한 배열 data를 사용하여 모든 원소를 3만큼 증가시키는 함수를 적절히 호출**
  - 그 결과 값을 출력하는 프로그램

Lab 9-2 — Lab2increment.c — 난이도: ★★

```
01  #include <stdio.h>
02
03  [          ]; //함수 increment의 함수원형
04  void printary(int data[], int SIZE); //함수 printary의 함수원형
05
06  int main(void)
07  {
08     int data[] = { 4, 7, 2, 3, 5 };
09     int aryLength = sizeof(data) / sizeof(int);
10     int inc = 3;
11
12     printary(data, aryLength); //배열 출력을 위해 printary() 함수 호출
13     //배열 원소를 모두 3씩 증가시키기 위해 increment() 함수 호출
14     [          ]
15     printf("배열 원소에 각각 %d의 더한 결과: \n", inc);
16     printary(data, aryLength); //결과를 알아 보기 printary() 함수 호출
17
18     return 0;
19  }
20
21  //배열크기가 SIZE인 배열 ary의 모든 원소를 n만큼 증가시키는 함수
22  void increment(int ary[], int n, int SIZE)
23  {
24     for (int i = 0; i < SIZE; i++)
25        [          ]
26  }
27
28  //배열크기가 SIZE인 배열 ary의 모든 원소를 출력하는 함수
29  void printary(int data[], int SIZE)
30  {
31     for (int i = 0; i < SIZE; i++)
32        printf("%2d ", data[i]);
33     printf("\n");
34  }
```

정답

```
03  void increment(int ary[], int n, int SIZE);
14     increment(data, inc, aryLength);
25        ary[i] += n;
```

재귀 함수와 재귀적 특성

- **재귀 함수(recursive function)**
  - 함수구현에서 자신의 함수를 호출하는 함수
- **재귀적 특성을 표현하는 알고리즘**
  - 재귀 함수를 이용하면 문제를 쉽게 해결

  - n의 계승(n factorial)을 나타내는 수식 n! 은 1 * 2 * 3 * . . . * (n-2) * (n-1) * n을 의미하는 수식이다. 즉 n!의 정의에서 보듯 계승은 n!을 정의하는 데 (n-1)!을 사용하는 재귀적 특성을 갖는다.

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

```
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));
}
```

n!을 구현한 재귀함수

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

실습예제 9-6 | Prj06 | 06factorial.c | n!을 구현한 재귀함수 | 난이도: ★

```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));
}
```

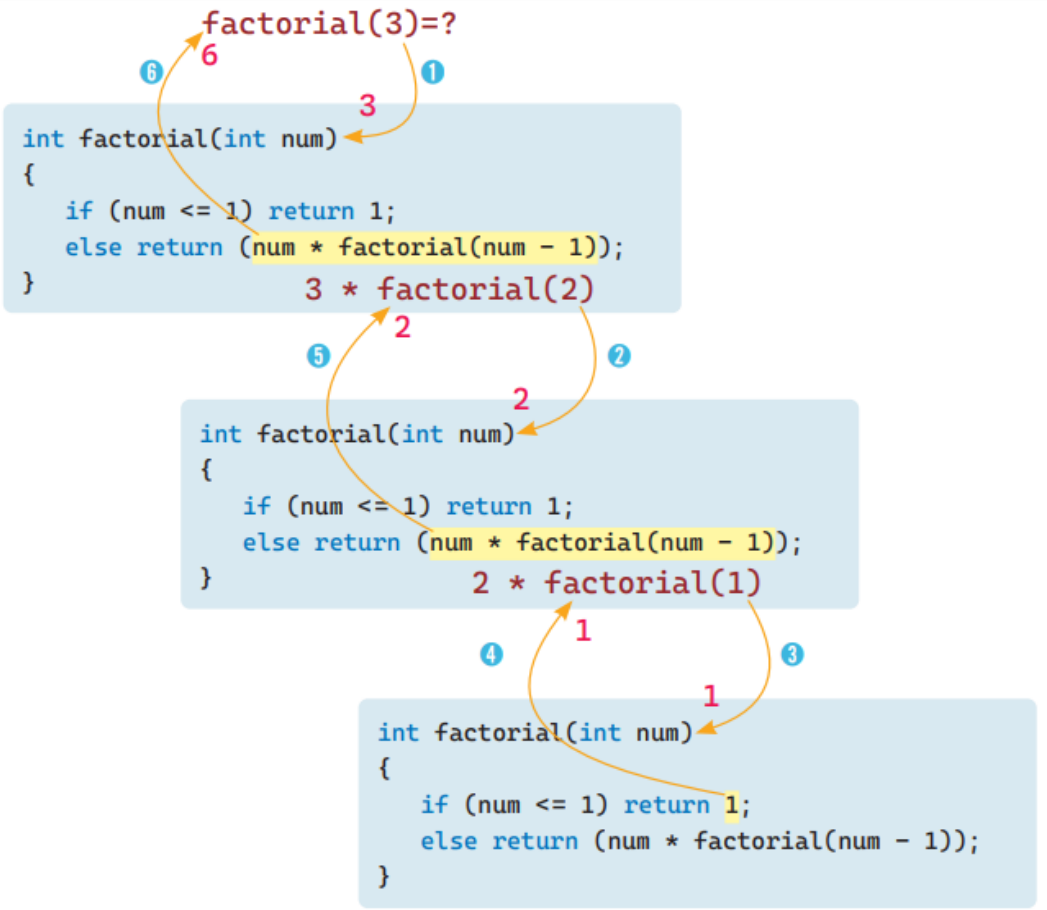
조건식을 만족하면 더 이상 자기 자신인 함수 factorial()을 호출하지 않는다.

결과

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

함수 factorial(3)을 호출한 경우 실행 과정

- **factorial(3)을 호출**
  - 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 반환 값 1 을 이용하여 return (2*1)을 실행**
  - 계속해서 factorial(3)으로 돌아와 factorial(2)의 결과값인 2를 이용하여 return (3*2)를 실행하면 결국 6을 반환
  - 즉 3!의 결과 6
- **3!을 구하기 위해 내부적으로 2번의 함수 호출**



난수와 구현

- **난수(random number)**
  - 특정한 나열 순서나 규칙을 가지지 않는 연속적인 임의의 수
    - **임의의 수란 어느 수가 반환될 지 예측할 수 없으며**
    - **어느 수가 선택될 확률이 모두 동일하다는 의미**
  - 주사위나 로또 복권 당첨기가 같은 기구가 난수를 발생시키는 장치
- **난수를 생성하는 함수 rand()를 이용**
  - 로또 프로그램 등 임의의 수가 필요로 하는 다양한 프로그래밍에 활용
  - 함수원형은 헤더파일 stdlib.h(standard library)에 정의
  - 0에서 32767사이의 정수 중에서 임의로 하나의 정수를 반환

실습예제 9-7 Prj07 07rand.c 난수를 위한 함수 rand()의 이용 난이도: ★

```
01  #include <stdio.h>
02  #include <stdlib.h> //rand()를 위한 헤더파일 포함
03
04  int main(void)
05  {
06     printf("0 ~ %5d 사이의 난수 8개: rand()\n", RAND_MAX);
07     for (int i = 0; i < 8; i++)
08        printf("%5d ", rand());
09     puts("");
10
11     return 0;
```

기호 상수 RAND_MAX를 16진수 0x7fff로 정의

결과

```
0 ~ 32767 사이의 난수 8개: rand()
   41 18467  6334 26500 19169 15724 11478 29358
```

여러 번 실행에도 항상 같은 수가 출력

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

- **함수 rand()는 함수호출 순서에 따라 항상 일정한 수가 반환**
  - 항상 41, 18467, 6334, 26500, 19169 등의 값들이 출력
- **매번 난수를 다르게 생성하려면**
  - 시드(seed) 값을 이용
    - **시드 값이란 난수를 다르게 만들기 위해 처음에 지정하는 수로서 시드 값이 다르면 난수가 달라짐**
- **srand(seed)를 호출**
  - 먼저 서로 다른 시드 값인 seed를 이용
    - **항상 서로 다른 seed 값을 지정하기 위해 현재 시간의 함수 time()을 이용**
  - 함수 time(NULL)은 1970년 1월 1일 이후 현재까지 경과된 시간을 초 단위로 반환

실습예제 9-8 Prj08 08srand.c srand()로 시드 값을 먼저 지정한 후 1에서 100 사이의 난수를 생성 난이도: ★

```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 8개:\n", MAX);
   for (int i = 0; i < 8; i++)
      printf("%5d ", rand() % MAX + 1);
   puts("");

   return 0;
}
```

함수 rand() 호출로 1에서 MAX 사이의 난수 출력

결과

```
1 ~   100 사이의 난수 8개:
   59    47    43    94    40    63    54     4
```

실행 시마다 다른 결과값이 출력

헤더파일 math.h

- 수학 관련 함수를 사용하려면 헤더파일 math.h을 삽입

Microsoft Visual Studio 디버그 콘솔

```
 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

2.72,  3.14,  9.00
4.00,  5.00, 10.20
```

실습예제 9-9 Prj09 09math.c 다양한 수학 관련 함수 난이도: ★

```
01  #include <stdio.h>
02  #include <math.h> //수학 관련 다양한 함수머리 포함 헤더파일
03
04  int main(void)
05  {
06     printf("  i    i제곱    i세제곱   제곱근(sqrt)\n");
07     printf("-----------------------------------\n");
08     for (int i = 3; i < 7; i++)
09        printf("%3d %7.1f %9.1f %9.1f\n", i, pow(i, 2), pow(i, 3), sqrt(i));
10     printf("\n");
11
12     printf("%5.2f, ", exp(1.0));
13     printf("%5.2f, ", pow(3.14, 1.0));
14     printf("%5.2f\n", sqrt(81));
15     printf("%5.2f, ", ceil(3.6));
16     printf("%5.2f, ", floor(5.8));
17     printf("%5.2f\n, ", fabs(-10.2));
18
19     return 0;
20  }
```

함수 sqrt(81)는 $\sqrt{81}$ 반환

함수 ceil(3.6)은 3.6의 천정 값인 4.0을 반환하는데, 천정 값 ceil(x)이란 x보다 작지 않은 가장 작은 정수를 뜻함

floor(5.8)은 5.8의 바닥 값인 5.0을 반환하는데, 바닥 값 floor(x)이란 x보다 크지 않은 가장 큰 정수를 뜻함

함수	
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(천정 값)
double floor(double x)	x보다 크지 않은 가장 큰 정수(바닥 값)
int abs(int x)	정수 x의 절대 값
double fabs(double x)	실수 x의 절대 값

헤더파일 ctype.h

- 문자 관련 함수는 헤더파일 ctype.h에 매크로로 정의

표 9-2 헤더파일 ctype.h의 주요 문자 관련 함수 매크로

함수원형	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 알파뉴메릭(alphanumeric)을 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)	아스키코드로 변환
_tolower(char)	무조건 영문 소문자로 변환
_toupper(char)	무조건 영문 대문자로 변환

실습예제 9-10 Prj10 10char.c 다양한 문자 관련 함수 난이도: ★

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03  #include <ctype.h> //문자 관련 함수는 헤더파일 ctype.h에 매크로로 정의
04
05  void print2char(char);
06
07  int main(void)
08  {
09     char ch;
10
11     printf("알파벳(종료x) 또는 다른 문자 입력하세요.\n");
12     do
13     {
14        printf("문자 입력 후 Enter: ");
15        scanf("%c", &ch);
16        getchar();                    //enter 키 입력 받음
17        if (isalpha(ch))
18           print2char(ch);
19        else
20           printf("입력: %c\n", ch);
21     } while (ch != 'x' && ch != 'X');  //입력이 x 또는 X이면 종료
22
23     return 0;
24  }
25
26  void print2char(char ch)
27  {
28     if (isupper(ch))
29        printf("입력: %c, 변환: %c\n", ch, tolower(ch));
30     else
31        printf("입력: %c, 변환: %c\n", ch, toupper(ch));
32
33     return;
34  }
```

문자 하나를 입력한 후 enter키를 반드시 누르도록 해 이 enter키를 하나 받아 들여 버리는 기능

입력 문자 ch가 영문자 알파벳 여부 검사

입력 문자가 영문자 알파벳이 아니면 바로 문자 그대로 출력

ch가 대문자이면 함수 tolower(ch)를 호출하여 소문자로 변환하여 출력

결과

```
알파벳(종료x) 또는 다른 문자 입력하세요.
문자 입력 후 Enter: w
입력: w, 변환: W
문자 입력 후 Enter: t
입력: t, 변환: T
문자 입력 후 Enter: @
입력: @
문자 입력 후 Enter: x
입력: x, 변환: X
```

다양한 라이브러리 함수를 제공

- **여러 헤더파일이 제공**
  - 여러 라이브러리 함수를 위한 함수원형과 상수, 그리고 매크로가 여러 헤더파일에 나뉘어 있음

표 9-3 여러 라이브러리를 위한 헤더파일

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, 난수, 메모리 할당 등) 함수

Lab 1에서 100사이의 난수 알아 맞추기

- 가장 먼저 난수를 생성시켜 변수 guess에 저장
- 입력 받은 정수를 input에 저장한 후
  - 저장된 guess와 비교하여 틀렸으면
    - **사용자에게 다음 입력을 위한 정보를 제공하고 다시 반복**
  - 입력한 정수 input과 guess가 같으면
    - **"정답입니다"를 출력하고 종료**

```
1에서 100 사이에서 한 정수가 결정되었습니다.
이 정수는 무엇일까요? 입력해 보세요. : 50
입력한 수 50보다 큽니다. 다시 입력하세요. : 75
입력한 수 75보다 작습니다. 다시 입력하세요. : 62
입력한 수 62보다 큽니다. 다시 입력하세요. : 69
입력한 수 69보다 작습니다. 다시 입력하세요. : 66
입력한 수 66보다 작습니다. 다시 입력하세요. : 64
정답입니다.
```

Lab 9-3 lab3numguess.c 난이도: ★★

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

정답

```
13     guess = rand() % MAX + 1 ;
26           break;
```

감사합니다.