프로그래밍 기초

9장. 함수 기초 Part 2

학습목표

- **주소에 의한 호출: Call by address**
  - 함수에서 배열의 주소 전달 및 함수 내부에서 배열의 값 변경

- **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 <stdlib.h>
    - 난수의 범위 지정 및 난수 초기화
  - 수학 관련 함수 이용: <math.h>
  - 문자 관련 함수 이용: <ctype.h>
    - 영문자 및 숫자 검사 및 대/소문자 변경

주소에 의한 호출 (Call by Address)

- **배열을 함수로 전달하는 방법**
  - C 언어는 배열 전체의 값을 함수로 전달할 수 없음

- **주소에 의한 호출(Call by Address)**
  - 함수에 변수(배열, 구조체 등)가 저장된 메모리주소를 전달하는 방식
    - 별도의 메모리 할당이 필요 없음
    - 원본 데이터와 메모리 주소를 공유
  - 함수 내부에서 원본 변수(배열)의 값을 직접 변경할 수 있음

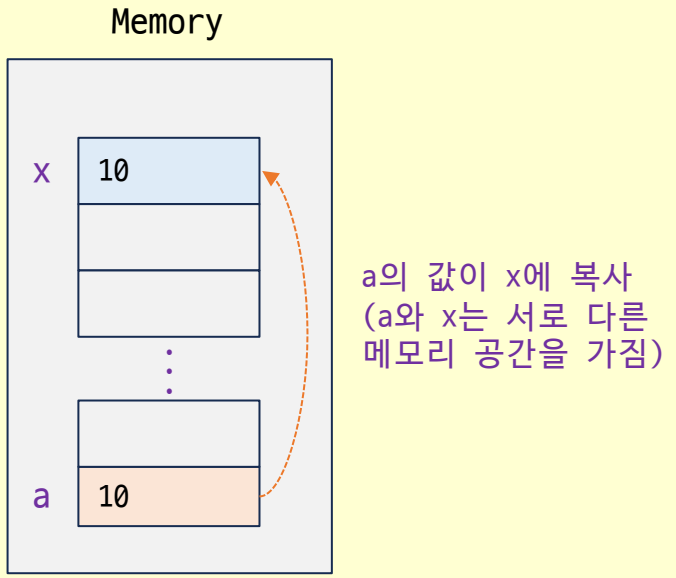
	값에 의한 호출 (Call by Value)	주소에 의한 호출 (Call by Address)
함수 호출	• 값을 전달	• 주소 전달
함수 내부에서 원본 데이터 변경 가능	• 변경 불가능	• 변경 가능

값에 의한 호출(Call by Value)

- 값에 의한 호출 예제

<callbyvalue.c>

```c
#include <stdio.h>

void func(int x)
{
    x = 100;
    printf("func 함수 x: %d\n", x);
}

int main()
{
    int a = 10;
    func(a);

    // a의 값은 변경 없음
    printf("main 함수 x: %d\n", a);
    return 0;
}
```



a의 값이 x에 복사
(a와 x는 서로 다른
메모리 공간을 가짐)

실행 결과

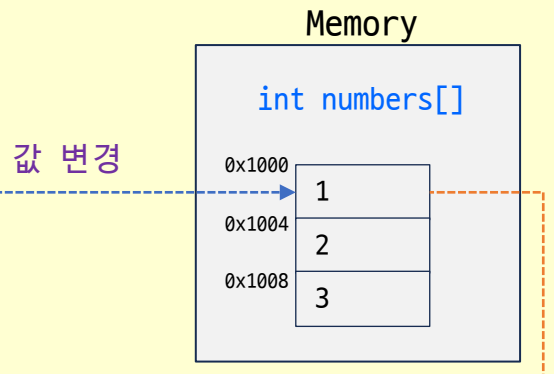
```
func 함수 x: 100
main 함수 x: 10
```

주소에 의한 호출(Call by Address)

- 주소에 의한 호출 예제

```c
#include <stdio.h>

void modify(int arr[], int n)
{ // int *arr 와 동일
    for (int i = 0; i < n; i++)
    {
        arr[i] += 10;
    }
}

int main()
{
    int numbers[3] = {1, 2, 3};
    // 배열의 시작 주소 전달 (Call by Address)
    modify(numbers, 3);

    for(int i = 0; i < 3; i++)
    {
        printf("%d ", numbers[i]); // 11 12 13 출력
    }
}
```

배열 numbers의 주소 전달
- arr[]과 numbers[]는 주소 공유



값 변경

값 읽기

실행 결과

```
11 12 13
```

LAB 배열의 모든 원소를 지정한 수만큼 증가시키는 함수 (실습)

- **함수 increment(int ary[], int n, int SIZE)**
  - 배열 크기가 SIZE인 배열 ary의 모든 원소를 n만큼 증가시키는 함수

<lab2increment.c>

```c
#include <stdio.h>

void increment(int ary[], int n, int SIZE);
void printary(int data[], int SIZE);

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

    printary(data, aryLength);

    //배열 원소를 모두 3씩 증가
    increment(data, inc, aryLength);

    printf("배열 원소에 각각 %d을 더한 결과: \n", inc);
    //결과를 알아 보기 printary() 함수호출
    printary(data, aryLength);

    return 0;
}
```

변경된 배열(data)의 값을 출력

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

원본 배열(data)의 값을 변경

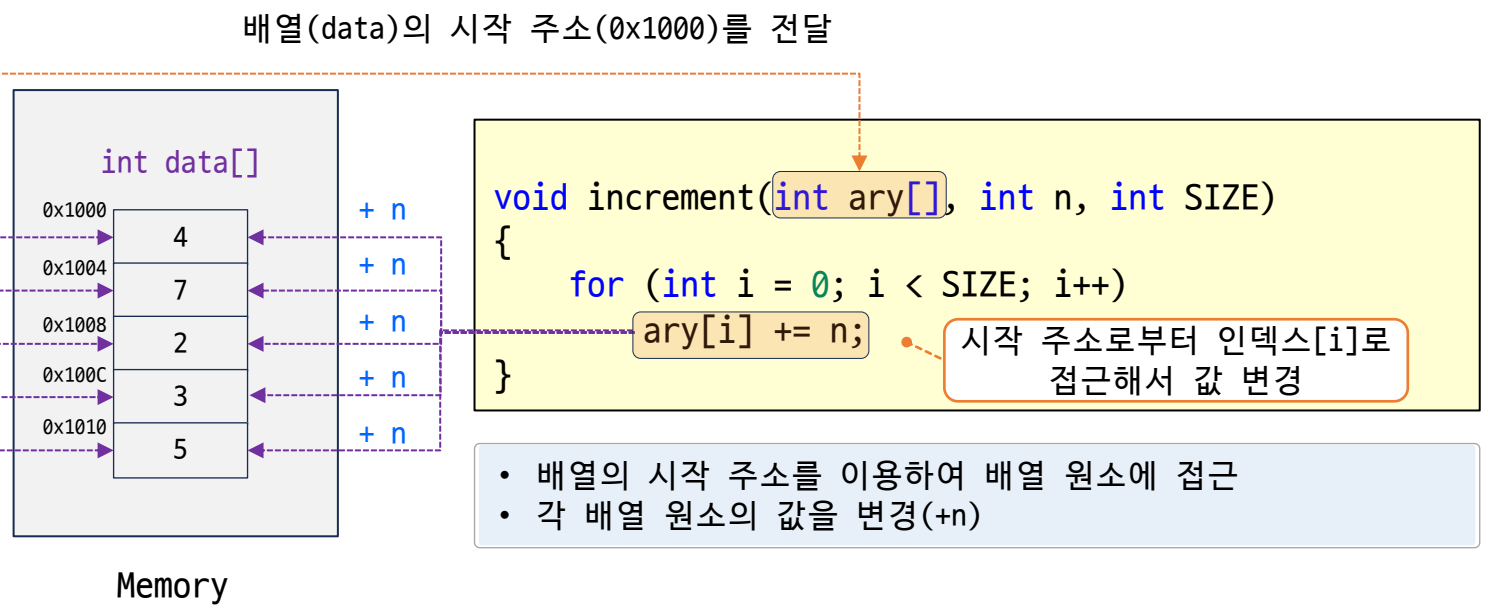
```c
//배열크기가 SIZE인 배열 ary의 모든 원소를 출력하는 함수
void printary(int data[], int SIZE)
{
    for (int i = 0; i < SIZE; i++)
        printf("%2d ", data[i]);
    printf("\n");
}
```

```
 4  7  2  3  5
배열 원소에 각각 3을 더한 결과:
 7 10  5  6  8
```

LAB increment() 함수 동작 과정

- **배열 변경 과정**
  - increment() 함수에서 변경한 배열의 값이 main()함수에도 적용됨: Call by address(C언어)
  - 배열 data[]의 주소에 저장된 값을 직접 변경하기 때문

```c
#include <stdio.h>

void increment(int ary[], int n, int SIZE);
void printary(int data[], int SIZE);

int main(void)
{
    int data[] = { 4, 7, 2, 3, 5 };
    int inc = 3;
            .  .  .

    //배열 원소를 모두 3씩 증가
    increment(data, inc, aryLength);

            .  .  .
    printary(data, aryLength);

    return 0;
}
```

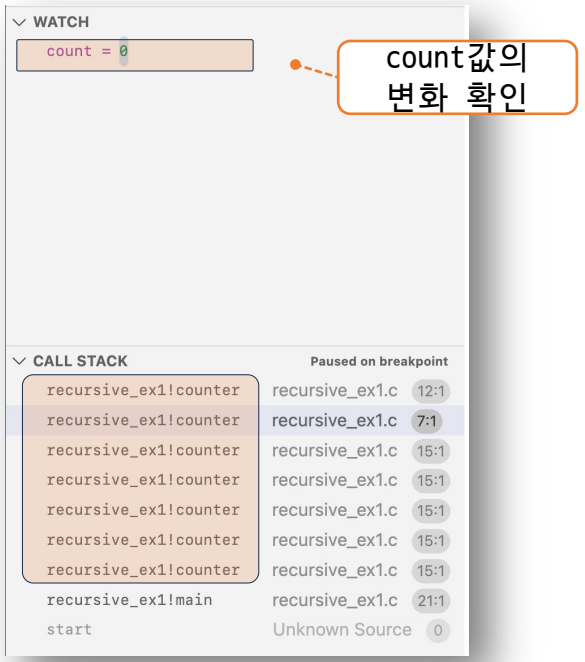


재귀 함수 정의 및 호출 (실습)

- **재귀 함수(recursive function)**
  - 함수 구현에서 자기 자신을 다시 호출하여 작업을 수행하는 함수
  - 해당 함수가 끝날 때까지 함수 호출 이후의 명령문이 수행되지 않음
  - 반드시 종료 조건이 포함되어야 함(무한 루프 방지)
    - ➢ Stack에 저장되기 때문에 종료 조건이 없으면 stack overflow 발생

<recursive_ex1.c>

```c
#include <stdio.h>

void counter(int count)
{
    if(count == 0) {
        printf("Count 종료");
        return;
    }

    counter(count-1); // 재귀 호출
    printf("Count: %d\n", count);
 }

int main() {
    counter(5);
    return 0;
}
```



VS Code

```
Counter 종료
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

재귀 함수와 재귀적 특성

- **재귀적 특성을 표현하는 알고리즘**
  - 재귀 함수를 이용하면 문제를 쉽게 해결

- **수학 수식에서 재귀적 특성**
  - n 계승(n factorial)을 나타내는 수식 n!
    - ➢ 1 * 2 * 3 * … * (n-2) * (n-1) * n을 의미
  - n!을 구하기 위해 (n-1)!을 먼저 구함
    - ➢ (n-1)!을 구하기 위해서는 다시 (n-2)!이 필요
  - n!을 함수 factorial(n)으로 구현한다면
    - ➢ 함수 factorial(n)에서 다시 factorial(n-1)을 호출

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

```
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 (실습)

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

<06factorial.c>

```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));
}
```

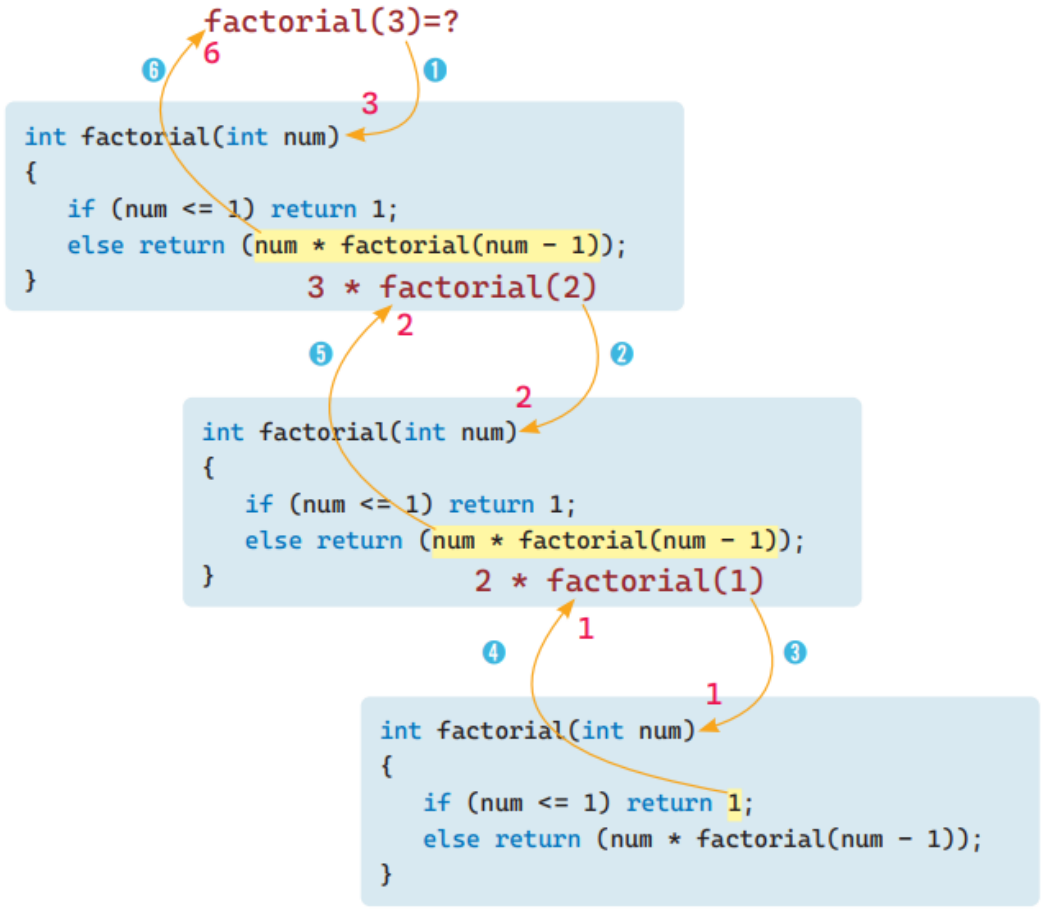
실행 결과

```
 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
  - 3!을 구하기 위해 내부적으로 2번의 함수 호출

- **재귀함수 단점**
  - 함수의 호출이 계속되면 시간이 오래 걸림
  - 메모리 사용이 많음

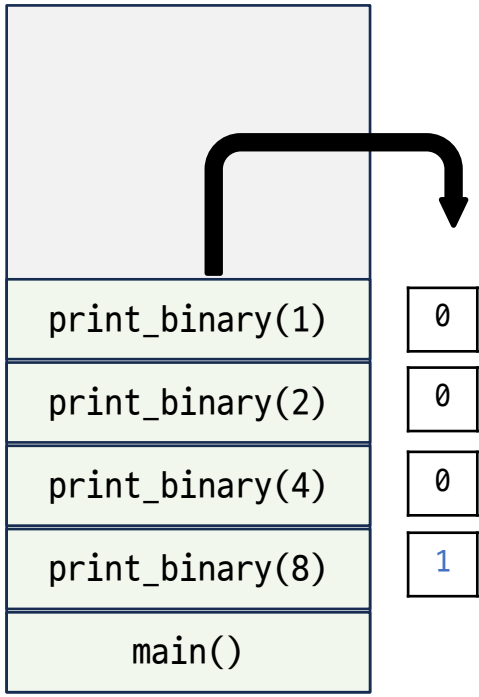


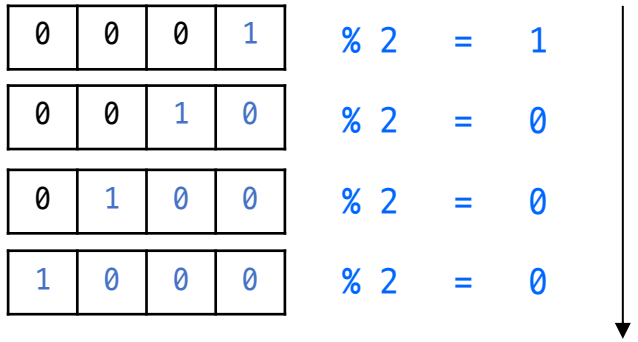
재귀함수 예제: 2진수 형식으로 출력하기

<06print_binary.c>

```c
#include <stdio.h>

void print_binary(int x);

int main()
{
    print_binary(8);
    printf("\n");
    return 0;
}

void print_binary(int x)
{
    if (x > 0)
    {
        // print_binary(x / 2);
        print_binary(x >> 1);
        printf("%d", x % 2);
    }
}
```

Call Stack





printf("%d", x % 2) 순서

% 2 = 1

% 2 = 0

% 2 = 0

% 2 = 0

실행 결과

```
1 0 0 0
```

난수(Random Number)와 구현

- **난수(random number)**
  - 특정한 나열 순서나 규칙을 가지지 않는 임의의 수
    - ➢ 임의의 수란 어느 수가 반환될 지 예측할 수 없으며, 어느 수가 선택될 확률이 모두 동일하다는 의미
  - 주사위나 로또 복권 당첨기가 난수를 발생시키는 장치

- **난수를 생성하는 함수 rand()를 이용**
  - 로또 프로그램 등 임의의 수가 필요로 하는 다양한 프로그래밍에 활용
  - 함수 원형은 stdlib.h(standard library)에 정의
  - RAND_MAX 값: Visual Studio(32767), GCC(214783647)

- **매번 난수를 다르게 생성하려면**
  - srand(seed)를 호출
    - ➢ 난수를 다르게 만들기 위해 처음에 지정하는 수
    - ➢ seed 값이 다르면 난수가 달라짐
  - srand(time(NULL))
    - ➢ 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

난수 출력

<07rand.c>

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

int main(void)
{
    printf("0 ~ %d 사이의 난수 8개: rand()\n", RAND_MAX);
    for (int i = 0; i < 8; i++)
        printf("%d ", rand());
    puts("");

    return 0;
}
```

RAND_MAX값
- GCC: 2147483647
- Visual Studio: 32767

GCC 실행 결과

```
0 ~ 2147483647 사이의 난수 8개: rand()
16807 282475249 1622650073 984943658 1144108930 470211272 101027544 1457850878
```

Visual Studio 실행 결과

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

srand() 예제: 1부터 100 사이의 난수 생성

<08srand.c>

```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)
{
    srand(time(NULL));

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

    return 0;
}
```

```
1 ~   100 사이의 난수 8개:
   28    46    98    90    40    62    12    40
```

rand()를 이용한 랜덤값 생성

- **생성할 랜덤값의 범위 지정**

start에서 end까지의 난수 발생	`(rand() % (end – start + 1)) + start`
1에서 n까지의 난수 발생	`(rand() % (n - 1 + 1)) + 1` `-> (rand() % n) + 1`

- **10 ~ 100까지 난수 발생**

- `(rand() % (100 – 10 + 1)) + 10`

1단계

```
rand() % (100 – 10 + 1)
rand() % 91
```

2단계

```
(rand() % 91) + 10
```

0			10
1			11
2			12
…	+ 10	=	…
88			98
89			99
90			100

rand()를 이용한 10~100랜덤값 생성 예제

<08srand2.c>

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
    int start = 10;
    int end = 100;
    srand(time(NULL));

    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10; j++)
            printf("%4d ", (rand() % (end - start + 1)) + start);
        printf("\n");
    }
    printf("\n");
}
```

```
  25   57   91   65   58   64   87   69   98   86
  63   28   40   33   65   36   78   36   93   76
  16   64   80   43   99   75   43   39   22   28
  95   86   94   57   22   42   73   85   42   90
  90   49   90   10   86   50   62   30   92   35
  44   34   85   12   88   40   35   87   56   57
  43  100   69   51   63   67   99   40   28   33
  89   56   16   51   85   75   63   86   90   54
  84   90   41   86   94   22   38   85   53   93
  18   86   82   88   35   95   93   36   78   82
```

수학 관련 함수

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

- 수학 관련 함수를 사용하려면 헤더파일 math.h을 삽입

<09math.c>

```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(3.14, 10): %5.2f, ", pow(3.14, 1.0));
    printf("sqrt(81): %5.2f\n", sqrt(81));
    printf("ceil(3.6): %5.2f, ", ceil(3.6));
    printf("floor(5.8): %5.2f, ", floor(5.8));
    printf("fabs(-10.2): %5.2f\n, ", fabs(-10.2));

    return 0;
}
```

실행 결과

```
  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(3.14, 10):  3.14, sqrt(81):  9.00
ceil(3.6):  4.00, floor(5.8):  5.00, fabs(-10.2): 10.20
```

round() 함수: 반올림 (실습)

- **double round(double x): 반올림된 실수형 값을 반환**
  - <math.h> 헤더 포함, 소수 점 자리 수 지정은 안됨

```c
#include <stdio.h>
#include <math.h>

int main()
{
    double a = 2.3;
    double b = 2.7;
    double c = -2.5;

    // 기본 반올림: printf("%.자리수f") 사용
    printf("round(%.2f) = %.2f\n", a, round(a)); // 출력: round(2.30) = 2.00
    printf("round(%.2f) = %.2f\n", b, round(b)); // 출력: round(2.70) = 3.00
    printf("round(%.2f) = %.2f\n", c, round(c)); // 출력: round(-2.50) = -3.00

    // 소수점 둘째 자리 반올림
    double pi = 3.141592;
    printf("round(pi*100)/100 = %.2f\n", round(pi * 100) / 100); // 출력: 3.14
    printf("round(pi*1000)/1000 = %.3f\n", round(pi * 1000) / 1000); // 출력: 3.142
    return 0;
}
```

반올림 소수점 자리수 지정 방법

실행 결과

```
round(2.30) = 2.00
round(2.70) = 3.00
round(-2.50) = -3.00
round(pi*100)/100 = 3.14
round(pi*1000)/1000 = 3.142
```

문자 관련 함수

- **문자 관련 함수**
  - 헤더 파일 ctype.h에 매크로로 정의
  - 문자를 검사하거나 문자를 변환하는 기능 수행
  - 문자 검사 함수: isxxx(char)
    - 문자 검사 함수는 0(false)과 0이 아닌 정수값(true)을 반환
  - 문자 변환 함수: toxxx(char)
    - 문자 변환 함수는 변환된 문자를 반환

- **주요 함수**
  - toupper(c)
    - c가 영문 소문자일 때 영문 대문자로 변환
  - tolower(c)
    - c가 영문 대문자일 때 영문 소문자로 변환

헤더파일 ctype.h: 문자관련 함수

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

다양한 문자 관련 함수 예제 (실습)

<10char.c>

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <ctype.h>

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 제거
        if (isalpha(ch))
            print2char(ch);
        else
            printf("입력: %c\n", ch);
    } while (ch != 'x' && ch != 'X'); //입력이 x 또는 X이면 종료

    return 0;
}
```

입력한 문자가 알파벳인지 검사

대문자 -> 소문자 변환
소문자 -> 대문자 변환

```c
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;
}
```

```
알파벳(종료x) 또는 다른 문자 입력하세요.
문자 입력 후 Enter:  a
입력: a, 변환: A
문자 입력 후 Enter: E
입력: E, 변환: e
문자 입력 후 Enter: w
입력: w, 변환: W
문자 입력 후 Enter: E
입력: E, 변환: e
문자 입력 후 Enter: 1
입력: 1
문자 입력 후 Enter: x
입력: x, 변환: X
```

다양한 라이브러리 함수를 제공

- **많은 라이브러리 함수 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 큽니다. 다시 입력하세요. : 90
입력한 수 90보다 큽니다. 다시 입력하세요. : 95
입력한 수 95보다 큽니다. 다시 입력하세요. : 96
입력한 수 96보다 큽니다. 다시 입력하세요. : 97
입력한 수 97보다 큽니다. 다시 입력하세요. : 98
정답입니다.
```

Lab 1에서 100사이의 난수 알아 맞추기

<lab3numguess.c>

```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)
{
    int guess, input;

    srand(time(NULL));
    guess = rand() % MAX + 1;

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

    while (scanf("%d", &input))
    {
        if (input > guess)
            printf("입력한 수 %d보다 작습니다. 다시 입력하세요. : ", input);
        else if (input < guess)
            printf("입력한 수 %d보다 큽니다. 다시 입력하세요. : ", input);
       else
       {
            puts("정답입니다.");
            break;
       }
    }
    return 0;
}
```

Questions?