PART 09 배열 연습문제 풀이

1. 배열 크기가 n이라면 최대 첨자는 몇인가?

n - 1이다.

2. 다음 중 C 언어의 배열에 대한 설명으로 옳지 않은 것은? ④

① 배열 이름은 변수가 아니라 상수이다.
② 배열은 선언하면서 초기화할 수 있다.
③ 배열 이름을 함수의 인수로 전달할 수 있다.
④ 배열을 선언할 때 배열 크기는 항상 명시해야 한다.

3. 다음 문장을 읽고 참, 거짓을 판별하라.

① 배열과 일반 변수는 함께 선언할 수 없다. (거짓. 함께 선언할 수 있다.)
② 배열 이름은 변수가 아니라 상수이다. (참. 배열 이름은 배열이 할당된 공간의 시작주소다.)
③ 배열을 함수의 인수로 전달할 때는 배열 이름과 크기를 별도의 인수로 전달한다. (참. 배열 크기를 별도의 인수로 전달하지 않으면 크기가 다른 배열을 처리할 수 없다.)
④ 배열을 선언할 때 배열 크기는 항상 명시해야 한다. (거짓. 초기화 목록이 주어진 경우에는 배열 크기를 생략할 수 있다.)
⑤ 2차원 배열도 1차원 배열 초기화 목록을 이용하여 초기화할 수 있다. (참. 2차원 배열도 1차원 배열로 구현되므로 1차원 배열 초기화 목록을 이용하여 초기화 할 수 있다.)

4. 날짜(연, 월, 일)를 입력받아서 그해에서 그 날이 몇 번째 날인지 출력하는 프로그램을 작성하라. 입력된 날짜가 올바른 날짜인지 먼저 판별해야 한다. 이 때, 해당 연도가 윤년인지 아닌지 판별하는 것이 중요하다. 왜냐하면 윤년이면 2월 29일을 허용해야 할 것이고 그렇지 않으면 허용해서는 안 되기 때문이다. 윤년 여부는 입력된 날짜가 3월 이후일 때에도 중요하다. 해당 날짜가 윤년인지 아닌지 판단하는 기준은 6장 연습문제 7번에 나타나 있으니 참고하기 바란다.

```c
#include <stdio.h>
#include <assert.h>
int is_leap_year(int year)
{
    return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
}
int day_of_year(int year, int month, int day)
{
    static int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int num_of_days = day;
    if (is_leap_year(year)) days[2]++;
    assert(num_of_days <= days[month]);
```

```
    for (int i = 0; i < month - 1; ++i)
        num_of_days += days[i];
    return num_of_days;
}
int main()
{
    int year, month, day;

    printf("Enter year, month, day: ");
    scanf("%d%d%d", &year, &month, &day);
    printf("%d/%d/%d is ");
    printf("%dth day of the year\n", day_of_year(year, month, day));
    return 0;
}
```

5. 다음 프로그램은 NUM 개의 정수를 입력으로 받아서 평균과 표준편차를 출력하는 프로그램이다.

```
01 //*************************************************************************
02 // avgStd.c
03 //
04 // NUM 개의 숫자를 읽은 후 평균과 표준편차를 구하는 프로그램
05 //*************************************************************************
06
07 #include <stdio.h>
08 #include <math.h>
09
10 #define NUM 10
11
12 int main()
13 {
14 float a[NUM];
15 double sum, avg, std;
16 int i;
17 // 숫자를 입력받음
18 printf("%d 개의 숫자를 차례로 입력하세요.\n", NUM);
19 for (i = 0; i < NUM; ++i)
20 scanf("%f", &(a[i]));
21
22 // 평균을 구함
23 sum = 0;
24 for (i = 0; i < NUM; ++i)
25 sum += a[i];
26 avg = sum / NUM;
27
```

```
28 // 표준편차를 구함
29 sum = 0;
30 for (i = 0; i < NUM; ++i)
31 sum += (a[i] - avg) * (a[i] - avg);
32 std = sqrt(sum / NUM);
33
34 // 평균과 표준편차를 출력함
35 printf("평균 = %lf\n", avg);
36 printf("표준편차 = %lf\n", std);
37
38 return 0;
39 }
```

위 프로그램에서 정수를 입력받는 부분, 평균을 계산하는 부분, 표준편차를 계산하는 부분을 모두 별도의 함수로 작성하라. 이들 함수는 모두 배열 인수를 사용하도록 해야 한다. 구체적으로 말해서 다음과 같은 프로토타입으로 함수를 작성하여라.

```
void read_array(int n[], int size);
double average(int n[], int size);
double std_dev(int n[], int size, double average);
```

(오타: 문제에 주어진 프로토타입에서 첫 번째 인수 int n[]은 float n[]으로 바꿔어야 함)

```
//*********************************************************************
// avgStd.c
//
// NUM 개의 숫자를 읽은 후 평균과 표준편차를 구하는 프로그램
//*********************************************************************

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

#define NUM 10

void read_array(float n[], int size);
double average(float n[], int size);
double std_dev(float n[], int size, double average);

void read_array(float n[], int size)
{
    int i;
    // 숫자를 입력받음
    printf("%d 개의 숫자를 차례로 입력하세요.\n", NUM);
    for (i = 0; i < size; ++i)
        scanf("%f", &(n[i]));
}
```

```
double average(float n[], int size)
{
    int i;
    double sum = 0;
    // 평균을 구함
    for (i = 0; i < size; ++i)
        sum += n[i];
    return sum / size;
}

double std_dev(float n[], int size, double average)
{
    int i;
    double sum = 0;
    // 표준편차를 구함
    for (i = 0; i < size; ++i)
        sum += (n[i] - average) * (n[i] - average);
    return sqrt(sum / size);
}

int main()
{
    float a[NUM];
    double avg, std;

    read_array(a, NUM);
    avg = average(a, NUM);
    std = std_dev(a, NUM, avg);

    // 평균과 표준편차를 출력함
    printf("평균 = %lf\n", avg);
    printf("표준편차 = %lf\n", std);

    return 0;
}
```

6. 소수(prime number)란 1과 자기 자신 외에 다른 약수가 없는 수를 말한다. 곱셈 항등원인 1은 소수 집합에서 제외한다. 에라토스테네스가 제안한 체로 거르는 방법(Eratosthenes' Sieve)으로 소수를 구할 수 있다. 1 이상의 자연수를 다음과 같이 나열하였을 때, 1은 소수가 아니므로 기본적으로 제외한다. 그 다음 수 2는 소수이다. 대신 3 이상의 수에서 2의 배수가 되는 것을 모두 지운다. 3은 남아 있으므로 소수이다. 대신 4 이상의 수에서 3의 배수가 되는 수를 모두 지운다. 다시 5는 이렇게 하여 지워지지 않고 남아 있으므로 소수이다. 대신 6 이상의 수에서 5의 배수가 되는 수를 모두 지운다. 이런 방식으로 반복하면 소수를 순서대로 구할 수 있다.

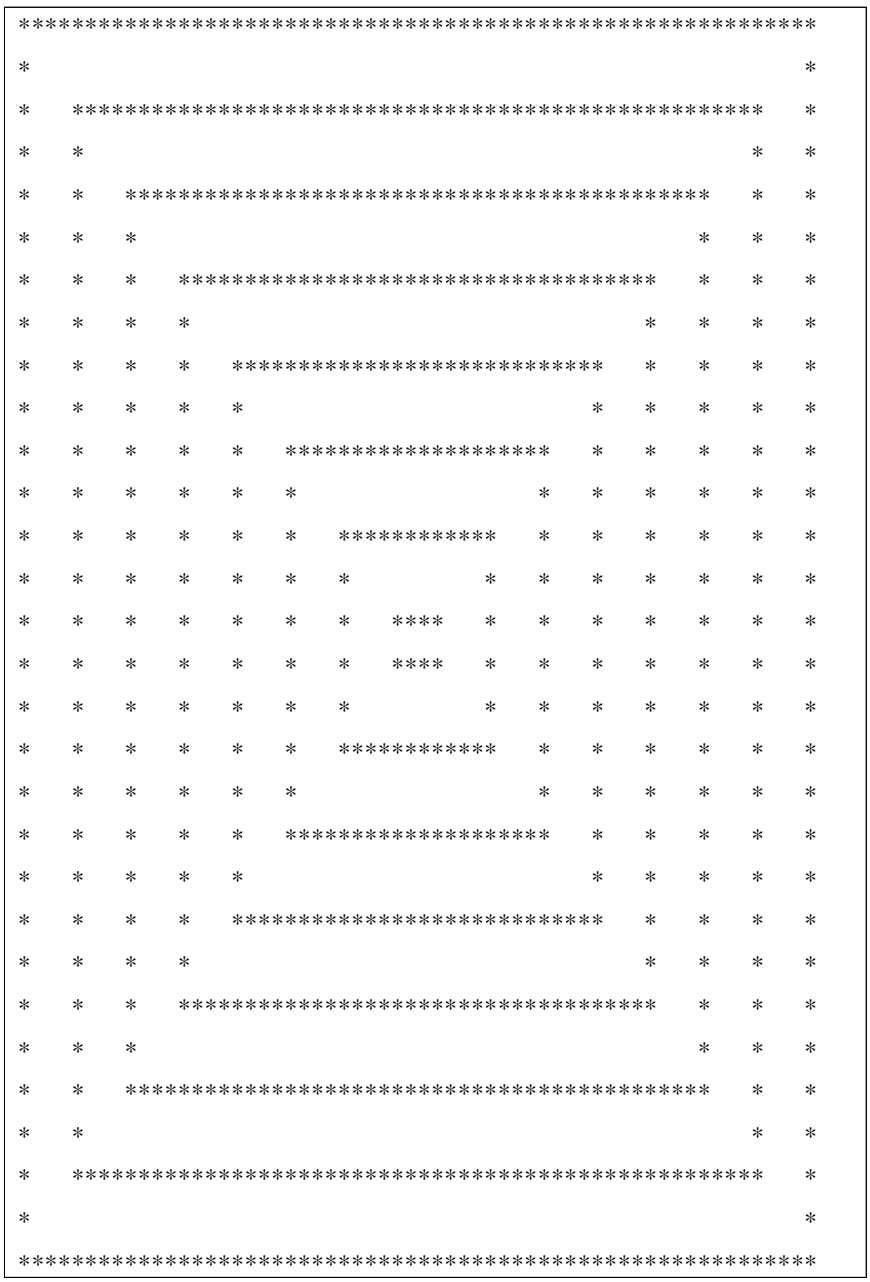
~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	...

에라토스테네스가 제안한 이 방법에 따라서 1에서 10,000 사이의 소수를 모두 구하라.

```
#include <stdio.h>

#define MAX 10000

void init_nums(int nums[], int max);
int find_next(int prev, int nums[], int max);
void sieve(int begin, int prime, int nums[], int max);

int main()
{
    int nums[MAX];
    int next_idx = 0;
    int count = 0;
    init_nums(nums, MAX);
    while ((next_idx = find_next(next_idx, nums, MAX)) != MAX) {
        int prime = nums[next_idx];
        printf("%5d", prime);
        sieve(next_idx, prime, nums, MAX);
        count++;
        if (count % 10 == 0) printf("\n");
    }
    if (count % 10 != 0) printf("\n");
    printf("%d primes are found\n", count);
    return 0;
}
void init_nums(int nums[], int max)
{
    int i;
    for (i = 0; i < max; ++i)
        nums[i] = i + 1;
    nums[0] = -1; // 1 is not a prime
}
int find_next(int prev, int nums[], int max)
{
    int i;
    for (i = prev + 1; i < max; ++i)
        if (nums[i] != -1) return i;
```

```
    return i;
}
void sieve(int begin, int prime, int nums[], int max)
{
    int i;
    for (i = begin + 1; i < max; ++i) {
        if (nums[i] % prime == 0) nums[i] = -1;
    }
}
```

7. char 형 2차원 배열 board[30][61]을 선언하라. 이 2차원 배열을 이용하여 다음과 같이 출력하는 프로그램을 작성하라.



```c
#include <stdio.h>

void init_board(char board[30][61])
{
    const char mark = '*', blank = ' ';
    int i, j;
    for (i = 0; i < 30; i++) {
        for (j = 0; j < 60; j++) {
            board[i][j] = mark;
        }
        board[i][j] = '\0';
    }
    for (i = 1; i < 15; i += 2) {
        for (j = i; j < 30 - i; j++) {
            // horizontal blanks
            board[i][2*j] = board[i][2*j+1] = blank;
            board[29-i][2*j] = board[29-i][2*j+1] = blank;
            // vertical blanks
            board[j][2*i-1] = board[j][2*i] = board[j][2*i+1] = blank;
            board[j][59-2*i-1] = board[j][59-2*i] = board[j][59-2*i+1] =
blank;
        }
    }
}

void print_board(char board[30][61])
{
    int i;
    for (i = 0; i < 30; i++)
        puts(board[i]);
}

int main()
{
    char board[30][61];
    init_board(board);
    print_board(board);
    return 0;
}
```

8. n개의 개체에 대해서 순열(permutation)이란 이들 개체를 일렬로 늘어놓은 것을 말한다. n개의 서로 다른 개체를 일렬로 늘어놓는 방법의 수는 n!이다. 정수 n을 입력으로 받아서 1~n 사이의 정수들에 대한 순열을 모두 구하는 프로그램을 작성하라. n은 1이상 7이하의 정수라고 가정한다.

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

#define MAX 7
#define MAXFACT 5040

int read_n();
void init_array(int array[], int size);
int fact(int n);
void perm(int array[MAX], int begin, int end, int perms[][MAX]);
void print_perms(int perms[][MAX], int end, int num);

int main()
{
    int n;
    int perms[MAXFACT][MAX] = {0};
    int init[MAX];

    n = read_n();
    init_array(init, n);
    perm(init, 0, n, perms);
    print_perms(perms, n, fact(n));

    return 0;
}

int read_n()
{
    int n;
    printf("Enter the number of objects n (0 < n < 8): ");
    scanf("%d", &n);
    assert(n >= 0 && n <= 7);
    return n;
}

void init_array(int array[], int size)
{
    int i;
    for (i = 0; i < size; ++i)
        array[i] = i + 1;
}

int fact(int n)
{
    assert(n >= 0);
    if (n == 0) return 1;
```

```
    return n * fact(n - 1);
}

void swap(int *p, int *q)
{
    int t = *p;
    *p = *q;
    *q = t;
}

void copy_array(int target[], int source[], int length)
{
    int i;
    for (i = 0; i < length; ++i)
        target[i] = source[i];
}

void perm(int array[MAX], int begin, int end, int perms[][MAX])
{
    int i, j, k = 0;
    int width, sub_size;
    int (*pivot)[MAX];
    assert(begin < end);
    width = end - begin;
    if (width == 1) {
        perms[0][begin] = array[begin];
        return;
    }
    sub_size = fact(width);
    // copy the begining part of 'sub_size' permutations
    for (i = 0; i < sub_size; ++i)
        for (j = 0; j < begin; ++j)
            perms[i][j] = array[j];
    // copying the array, setting the begin index, and recursing
    sub_size /= width;
    for (i = 0; i < width; ++i) {
        int array2[MAX];
        copy_array(array2, array, end);
        swap(&array2[begin], &array2[begin+i]);
        pivot = &perms[k];
        for (j = 0; j < sub_size; ++j)
            perms[k++][begin] = array2[begin];
        perm(array2, begin + 1, end, pivot);
    }
}
```

```
void print_perms(int perms[][MAX], int end, int num)
{
    int i, j;
    for (i = 0; i < num; ++i) {
        printf("%4d: ", i + 1);
        for (j = 0; j < end; ++j)
            printf("%d", perms[i][j]);
        printf("\n");
    }
}
```