프로그래밍 기초

8장. 배열

학습목표

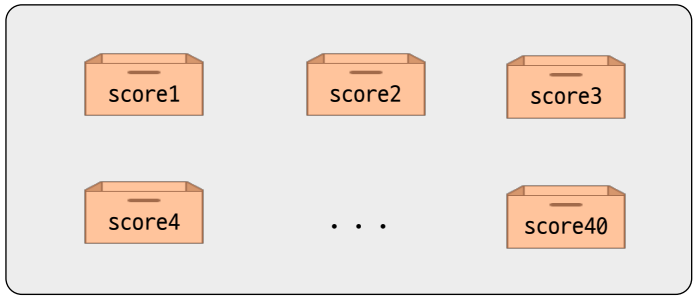
- **배열의 개요와 배열 선언 구문에 대하여 이해하고 설명할 수 있다.**
  - 여러 자료의 처리에 필요한 자료구조
  - 자료형, 배열이름, 배열크기를 이용한 배열 선언
  - 생성된 배열에서 원하는 원소를 참조
  - 배열 선언 시 동시에 초기값 지정 방법
  - 배열 선언 초기값 설정에서 배열크기 관계
  - 배열에서의 기본값과 쓰레기 값

- **다차원 배열에 대하여 다음을 이해하고 설명할 수 있다.**
  - 2차원 배열의 개념과 선언 방법
  - 2차원 배열의 배열 선언 초기값 설정
  - 3차원 배열의 개념과 배열 선언과 초기값 설정

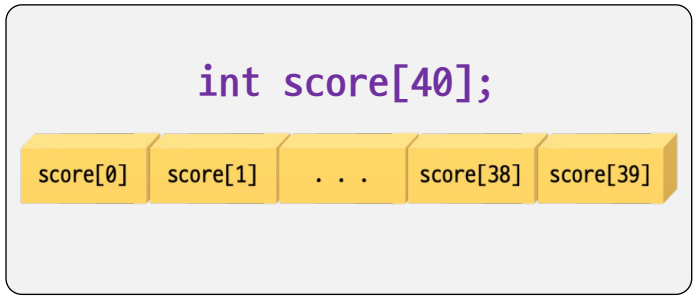
배열의 정의와 필요성

- **배열(array)의 필요성**
  - 40명 학생 성적을 변수에 저장하려면 40개의 변수를 선언
  - 여러 변수들이 같은 이름으로 연속된 메모리에 저장되는 구조가 있다면 편리: 배열
    - 변수를 일일이 선언하는 번거로움을 해소
    - 배열을 구성하는 각각의 변수를 참조하는 방법이 간편: index

- **배열의 정의**
  - 한 자료 유형의 원소를 지정된 크기만큼 확보한 연속된 저장 공간
    - 배열 원소(element): 배열을 구성하는 각각의 항목
      - 배열 원소 접근: 인덱스(index) 번호라는 숫자를 이용하여 쉽게 접근
  - 배열의 구성: 자료 유형, 배열 이름, 배열 크기



일반 변수의 활용



배열의 활용

배열 선언

- **배열 선언**
  - 자료형 배열이름 [배열크기];

```
int  data[10];
```

자료형 배열이름 배열크기

➢ 양의 정수와 매크로 상수 사용

➢ 변수와 const 상수로는 배열의 크기 지정이 불가능

```
#define SIZE 5

int score[10];
double point[20];
char ch[80];
int score[SIZE + 1];
```

정상 배열 선언

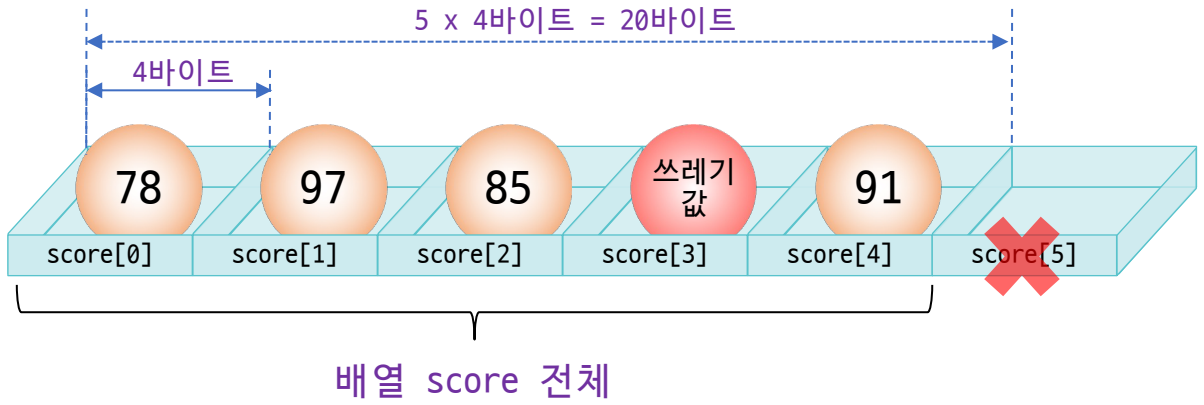
```
int n = 5;

int score[n];       // 변수 사용
double point[-3];   // 음수 사용
float grade[3.2];   // 실수 사용
int score[n + 2];   // 변수 연산 사용
```

잘못된 배열 선언

배열 원소 접근

- **인덱스(index)를 이용**
  - 배열 이름 뒤에 대괄호 사이: array[index]
    - 첫 번째 배열원소를 참조하는 인덱 값: 0, 다음 두 번째 원소는 1
    - 배열 인덱스의 범위: 0 ~ (배열크기-1)까지
      - array[0], array[1], array[2] … array[크기-1]
    - 인덱스의 유효 범위를 벗어난 원소를 참조하면 실행 오류 발생

```
int score[5];

score[0] = 78;  //ddd
score[1] = 97;
score[2] = 85;
// score[3]에 값 저장하지 않음: 쓰레기값 저장
score[4] = 91;
score[5] = 50; // 실행 오류 발생
```



배열 원소 일괄 출력 (실습)

- **배열 선언 후 배열 원소에 값을 저장하고 순차적으로 출력**

<01decarray.c>

```c
#include <stdio.h>
#define SIZE 5

int main(void)
{
    // 배열선언
    int score[SIZE]; // int score[5];

    // 배열 원소에 값 저장
    score[0] = 78; // 인덱스를 사용해서 배열원소에 값을 저장
    score[1] = 97;
    score[2] = 85;
    // 배열 4번째 원소 score[3]에 값을 대입하지 않아 쓰레기 값 저장
    score[4] = 91;
    // score[5] = 50; // 문법오류 발생

    // 배열원소 출력
    for (int i = 0; i < SIZE; i++)
        printf("%d ", score[i]);
    printf("\n");

    return 0;
}
```

배열 원소 출력을 위한 반복 구분

score[3]은 초기화가 되지 않아서 쓰레기 값 저장

실행 결과 #1

```
78  97  85  1  91
```

배열의 크기(5)를 넘는 접근[5] -> 쓰레기 값 출력

```c
for (int i = 0; i <= SIZE; i++)
    printf("%d ", score[i]);
```

실행 결과 #2

```
78  97  85  82575360  91  -2101215035
```

배열 초기화

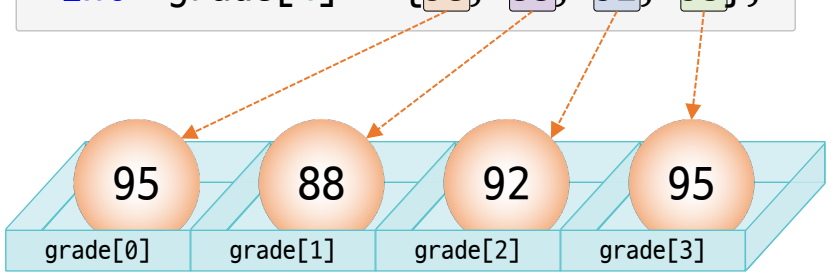
- **배열 선언 및 초기화**
  - 배열을 선언하면서 동시에 원소 값을 손쉽게 저장하는 (initialization) 방법
  - 중괄호 사이에 여러 원소 값을 쉼표로 구분하여 기술하는 방법
    - ➢ 중괄호 사이에는 명시된 배열 크기를 넘지 않게 원소 값 나열 가능
  - 배열 크기는 생략 가능: 일반적인 선언 및 초기화 방법
    - ➢ 배열 크기를 생략하면 자동으로 중괄호 사이에 입력된 원소 수가 배열 크기
    - ➢ `int data[] = {1, 2, 3, 4, 5};` → 배열의 크기는 자동으로 5가 됨

```
원소자료형  배열이름[배열크기] = {원소값1, 원소값2, … 원소값n};
원소자료형  배열이름[] = {원소값1, 원소값2, … 원소값n};
```

배열 크기를 생략하면 원소값의 수가 배열의 크기가 됨

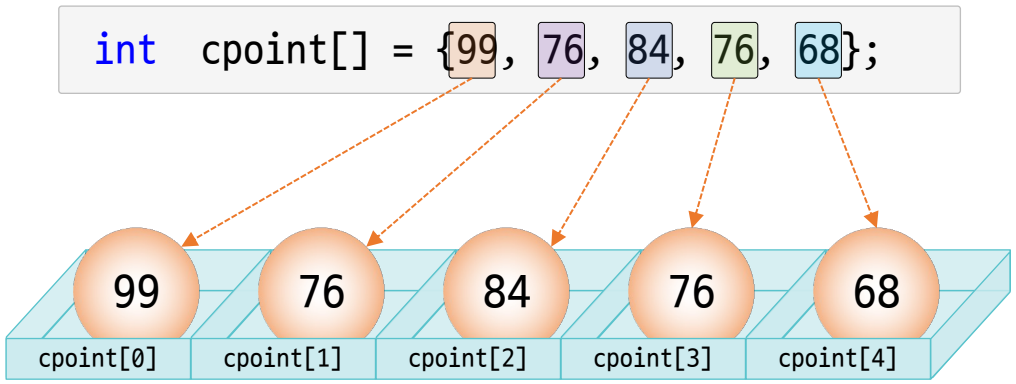
```
int  grade[4] = {98, 88, 92, 95};
double  output[] = {78.4, 90.2, 32.3, 44.6, 59.7, 98.9};
int  cpoint[] = {99, 76, 84, 76, 68};
```

```
int  grade[4] = {98, 88, 92, 95};
```

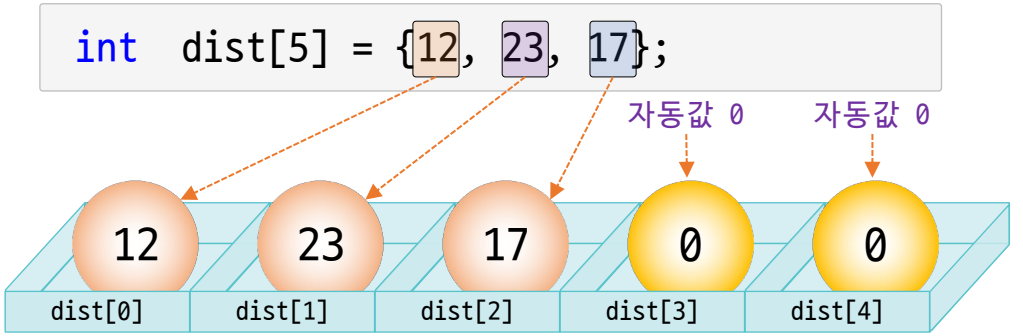


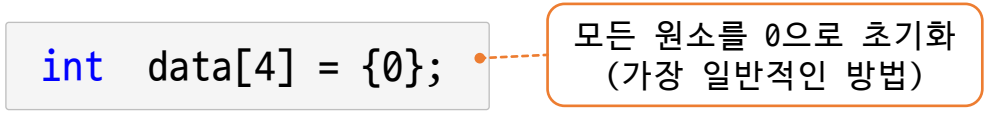
배열 기본 값

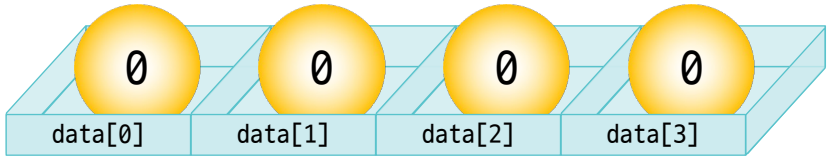
- **배열 크기가 초기값 원소 수보다 크면**
  - 지정하지 않은 원소의 초기값은 자동으로 모두 기본값이 저장
    - 기본값이란 자료형에 맞는 0
      - 정수형은 0, 실수형은 0.0 그리고 문자형은 '\0'인 NULL 문자
- **배열 크기가 초기값 원소 수보다 작으면**
  - 배열 공간을 벗어나므로 문법 오류 발생: warning: excess elements in array initializer



배열의 크기를 지정하지 않음: 원소수가 배열의 크기가 됨







배열 선언 초기화를 이용한 합과 평균 출력 (실습)

<02initarray.c>

```c
#include <stdio.h>
#define SIZE 6

int main(void)
{
    //배열 score의 선언과 초기화
    double score[] = { 89.3, 79.2, 84.83, 76.8, 92.52, 97.4 };
    double sum = 0;

    //for 문을 이용하여 합을 구함
    for (int i = 0; i < SIZE; i++)
    {
        sum += score[i];
        printf("score[%d] = %.2f\n", i, score[i]);
    }
    printf("성적의 합은 %.2f이고 평균은 %.2f이다.\n", sum, sum / SIZE);

    return 0;
}
```

```c
double score[6];
score = {89.3, 79.2, 84.83, 76.8, 92.52, 97.4};
```

중괄호를 사용한 초기화 방법
- 배열 선언시에 사용 가능
- 배열 선언과 값 초기화가 분리되어 있음
- 컴파일 에러 발생

실행 결과

```
score[0] = 89.30
score[1] = 79.20
score[2] = 84.83
score[3] = 76.80
score[4] = 92.52
score[5] = 97.40
성적의 합은 520.05이고 평균은 86.67이다.
```

다양한 배열 선언 초기화 구문

- **바른 문장과 잘못된 초기화 문장**

올바른 초기화 문장

```
#define SIZE 3

int grade[4] = {98, 88, 92, 95};
double output[SIZE] = {8.4, 0.2, 2.3, 44.6};
int cpoint[] = {99, 76, 84, 76, 68, 93};
char ch[] = {'a', 'b', 'c'};
double width[4] = {23.5, 32.1};
```

TIP 배열 선언 초기화에서 주의점

다음은 오류가 발생하는 배열 선언 초기화 문장이다. 초기화에서도 변수와 const 상수는 배열크기로 사용할 수 없다. 마지막 문장은 초보자가 자주 실수하는 문장으로, 중괄호를 사용한 **초기화 방법은 반드시 배열 선언 시에만 이용이 가능하며 배열 선언 이후에는 사용할 수 없으니 주의하자.**

표 8-2 배열 선언 초기화 시 오류 발생과 원인

변수와 배열 선언 문장	설명 및 오류 원인
`int n = 5;` `const int size = 6;`	변수 n과 const 상수 size 선언
`int score[n] = {89, 92, 91};` `int cpoint[size] = {3, 5, 7};`	변수 n은 배열크기로 사용 불가 상수 변수 size는 배열크기로 사용 불가
`int grade[3] = {98, 88, 92, 95};`	원소 수 4가 배열크기 3보다 큼
`int cpoint[] = {99 76 84 76 68 93};`	원소값을 구분하는 콤마(,)가 빠짐
`char ch[] = {a, b, c};`	원소값인 a, b, c가 문자여야 함
`double width[4];` `width = {23.5, 32.1};`	배열 선언 이후에는 중괄호를 사용한 초기화를 사용할 수 없으며, 배열 선언 시 double width[4] = {23.5, 32.1};로는 가능

NOTE: **C99: 배열의 첨자 초기화(designated initializers)**

배열의 초기화 방법이 다음과 같이 첨자를 사용해 부분적으로 초기값을 지정할 수 있다. 배열의 크기가 지정된 배열 a에서 지정한 첨자에 대해서는 초기값이, 그 외의 원소는 0으로 지정된다. 배열의 크기가 지정되지 않은 배열 b에서 지정한 첨자에 대해서는 초기값이, 그 외의 원소는 0으로 지정되며, 가장 큰 첨자가 마지막 원소가 되어 배열의 크기가 결정된다. 일반적인 배열 초기화 방법인 배열 c에서 순서대로 초기값이 저장되며 지정한 첨자에 대해서는 초기값이, 그 외의 원소는 0으로 저장된다.

```
int a[8] = { [1] = 10, [3] = 30, [5] = 50 };  // 0 10 0 30 0 50 0 0 저장
int b[] = { [1] = 10, [3] = 30, [5] = 50 };   // 0 10 0 30 0 50 저장
int c[] = { 1, 2, [2] = 10, [5] = 50 };       // 1 2 10 0 0 50 저장
```

문자(char) 배열: 실습

<chararray.c>

```c
#include <stdio.h>

int main(void)
{
    // 1. 문자 저장용 char형 배열 선언과 초기화
    char alpha[5] = {'A', 'B', 'C', 'D', 'E'};
    int i;

    // 2. 배열 요소 출력 (한 글자씩)
    printf("한 글자씩 출력: ");
    for (i = 0; i < 5; i++)
    {
        printf("%c ", alpha[i]);
    }
    printf("\n");

    // 3. 문자열로 활용 (문자열 끝에 NULL 문자 필요)
    char str[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("문자열 출력: %s\n", str);

    return 0;
}
```

실행 결과

```
한 글자씩 출력: A B C D E
문자열 출력: Hello
```

Lab 정수형(int) 배열에 표준입력으로 받은 정수를 저장하여 출력

- **표준입력으로 받은 여러 정수를 순서대로 배열 input에 저장하여 출력**

<lab1inputarray.c>

```c
#include <stdio.h>

int main(void)
{
    //초기화로 모든 원소에 0을 저장
    int input[20] = {0};

    printf("배열에 저장할 정수를 여러 개 입력하시오.\n");
    printf("0을 입력하면 입력을 종료합니다.\n");
    int i = 0;
    do {
        scanf("%d", &input[i]);
    } while (input[i++] != 0);

    i = 0;
    while (input[i] != 0) {
        printf("%d ", input[i++]);
    }
    puts("");

    return 0;
}
```

표준 입력으로 받은 정수는 0이 입력될 때까지 저장

실행 결과

```
배열에 저장할 정수를 여러 개 입력하시오.
0을 입력하면 입력을 종료합니다.
10
20
30
40
50
0
10 20 30 40 50
```

배열 복사

- **배열 복사**
  - 각각의 원소를 하나씩 다른 배열에 저장(복사)해야 됨

```
int a[5] = {1, 2, 3, 4, 5};
int b[5] = {0};


// 배열 복사: 오류
b = a;
```


```
arraycomp.c:9:11: error: assignment to
expression with array type
    9 |         b = a;
```

<arraycopy.c>

```
#include <stdio.h>

int main()
{
    int a[5] = {1, 2, 3, 4, 5};
    int b[5] = {0};

    for (int i = 0; i < 5; i++)
    {
        b[i] = a[i];
        printf("b[%d]: %d\n", i, b[i]);
    }

    return 0;
}
```

배열 a의 각 원소를 하나씩 복사

실행 결과

```
b[0]: 1
b[1]: 2
b[2]: 3
b[3]: 4
b[4]: 5
```

배열 비교

- **두 배열 비교**
  - 각 원소의 값을 하나씩 비교

```c
#include <stdio.h>

int main()
{
    int a[5] = {1, 2, 3, 4, 5};
    int b[5] = {1, 2, 3, 4, 5};

    if(a == b)
        printf("배열 a와 b는 같습니다.\n");
    else
        printf("배열 a와 b는 다릅니다.\n");
}
```

실행 결과

```
배열 a와 b는 다릅니다.
```

<arraycomp.c>

```c
#include <stdio.h>

int main()
{
    int a[5] = {1, 2, 3, 4, 5};
    int b[5] = {6, 2, 3, 4, 5};
    int equal = 1;

    for (int i = 0; i < 5; i++)
    {
        if(a[i] != b[i])
        {
            equal = 0;
            break;
        }
    }

    if(equal == 1)
        printf("배열 a와 b는 같습니다.\n");
    else
        printf("배열 a와 b는 다릅니다.\n");

    return 0;
}
```

두 배열에서 같은 위치의 값들을 모두 비교

배열의 최대값, 최소값

- **배열 원소 중 최대값, 최소값 찾기**
  - 최대값 비교
    - 가장 작은 값을 기준값(시작값)으로 설정
    - `int max = 0;`
  - 최소값 비교
    - 가장 큰 값을 기준값(시작값)으로 설정
    - `int min = 10000;`

```c
#include <stdio.h>
#define SIZE 8

int main()
{
    int price[SIZE] = {9000, 2700, 9800, 3500, 8500, 2300, 4000, 5700};
    int max = 0, min = 10000;
    int max_index = 0, min_index = 0;

    for (int i = 0; i < SIZE; i++)
    {
        if(price[i] >= max) // 최대값(max) 구하기
        {
            max = price[i];
            max_index = i;
        }
        if(price[i] <= min) // 최소값(min) 구하기
        {
            min = price[i];
            min_index = i;
        }
    }
    printf("max: %d, max_index: %d\n", max, max_index);
    printf("min: %d, min_index: %d\n", min, min_index);
    return 0;
}
```

최대값 업데이트

최소값 업데이트

실행 결과

```
max: 9800, max_index: 2
min: 2300, min_index: 5
```

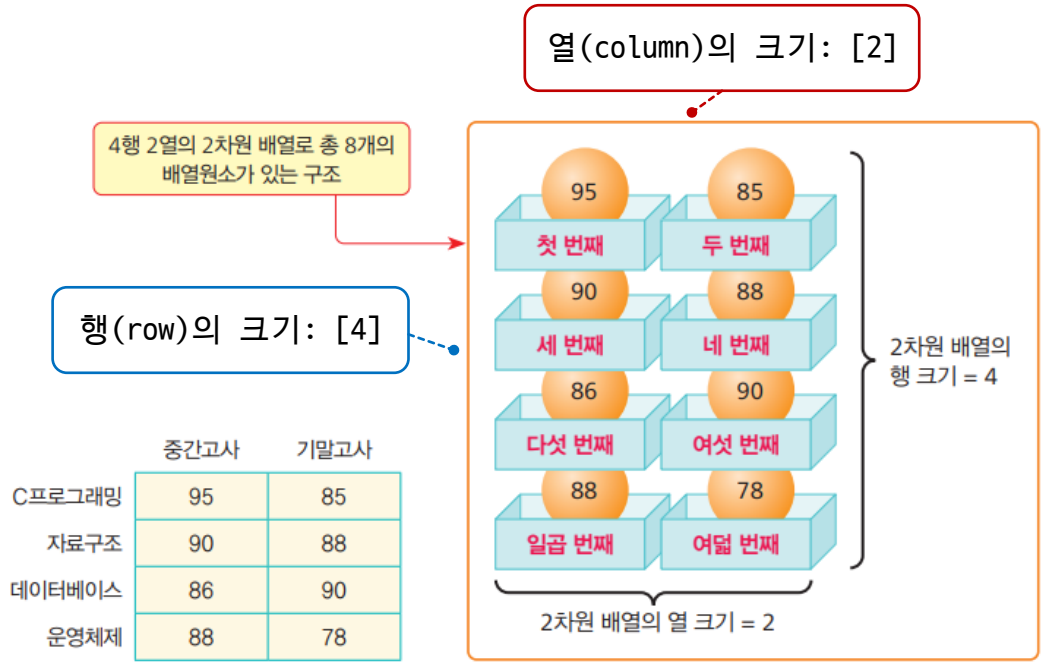
2차원 배열 개요

- **2차원 배열은 테이블 형태의 구조**
  - 행(row)과 열(column)의 구조로 표현

```
int score[ROW][COL];
```

  - ➢ 행(row) 우선 저장 방식

- **2차원 배열 선언**
  - 2개의 대괄호('[][]')가 필요
  - 첫 번째 대괄호: 행(row)의 크기
  - 두 번째 대괄호: 열(column)의 크기
  - 2차원 배열 선언 시 초기값을 지정하지 않으면
    - ➢ 반드시 행과 열의 크기를 명시해야 됨



	중간고사	기말고사
C프로그래밍	95	85
자료구조	90	88
데이터베이스	86	90
운영체제	88	78

2차원 배열 선언

원소자료형 배열이름[배열행크기][배열열크기];

배열 선언 시 배열크기는 생략할 수 없으며 배열크기는 리터럴 상수, 매크로 상수 또는 그들의 연산식이 허용된다.

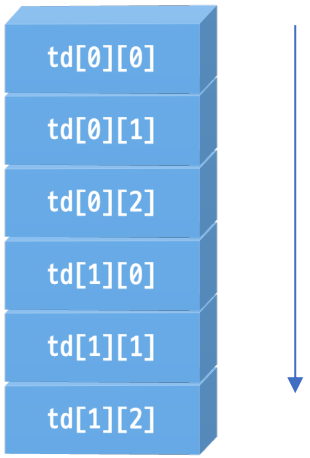
```
#define RSIZE 5
#define CSIZE 2

int score[RSIZE][CSIZE];

double point[2][3];
char ch[5][80];
float grade[7][CSIZE];
```

2차원 배열 구조

- **배열 선언 int td[2][3];**
  - td[0][0]으로 첫 번째 원소를 참조
    - 두 번째 원소는 td[0][1],
      - 두 번째 행의 첫 원소(4번째 원소): td[1][0]으로 행(row) index가 1 증가
    - 행(row) index: 0 ~ (행의 크기 - 1)
    - 열(column) index: 0에서 (열의 크기 - 1)
- **행 우선 (row major) 배열**
  - 행을 먼저 배치하는 특징
    - 첫 번째 행의 모든 원소가 메모리에 할당된 이후
    - 두 번째 행의 원소가 순차적으로 할당

```
#define ROWSIZE 2
#define COLSIZE 3

// 2차원 배열 선언
int td[ROWSIZE][COLSIZE];

// 2차원 배열 원소에 값 저장
td[0][0] = 1; td[0][1] = 2; td[0][2] = 3;
td[1][0] = 4; td[1][1] = 5; td[1][2] = 6;
```

실제 메모리 저장
(행 우선 저장)



2차원 배열 원소 참조

- **외부 반복 제어 변수 i**
  - 행을 0에서 (행의 수-1)까지 순차적으로 참조
- **내부 반복 제어 변수 j**
  - 0에서 (열의 수-1)까지 열을 순차적으로 참조

외부 반복 제어변수(i)는 행을 순차적으로 참조

```
for (i = 0; i < ROWSIZE; i++)
{
    for (j = 0; j < COLSIZE; j++)
        printf("%d ", td[i][j]);
    puts("");
}
```

내부 반복 제어변수(j)는 열을 순차적으로 참조

- **배열 원소의 저장 값이 행과 열의 관계식이 가능**
  - 다음 코드와 같이 중복된 반복문을 사용하여 값을 저장

```
for (i = 0; i < ROWSIZE; i++)
    for (j = 0; j < COLSIZE; j++)
        td[i][j] = (i * COLSIZE) + (j + 1);
```

```
td[0][0] = 1 td[0][1] = 2 td[0][2] = 3
td[1][0] = 4 td[1][1] = 5 td[1][2] = 6
```

2차원 배열 선언과 직접 초기값 저장 (실습)

<03tdarray.c>

```c
#include <stdio.h>

#define ROWSIZE 2
#define COLSIZE 3

int main(void)
{
    // 이차원 배열선언
    int td[ROWSIZE][COLSIZE];

    // 이차원 배열원소에 값 저장
    td[0][0] = 1; td[0][1] = 2; td[0][2] = 3;
    td[1][0] = 4; td[1][1] = 5; td[1][2] = 6;

    printf("반목문 for를 이용하여 출력\n");
    for (int i = 0; i < ROWSIZE; i++)
    {
        for (int j = 0; j < COLSIZE; j++)
            printf("td[%d][%d] == %d ", i, j, td[i][j]);
        printf("\n"); //행마다 한 줄 출력 후 다음 줄로 이동
    }
    return 0;
}
```

```c
for (i = 0; i < ROWSIZE; i++)
    for (j = 0; j < COLSIZE; j++)
        td[i][j] = (i * COLSIZE) + (j + 1);
```

실행 결과

```
반목문 for를 이용하여 출력
td[0][0] == 1 td[0][1] == 2 td[0][2] == 3
td[1][0] == 4 td[1][1] == 5 td[1][2] == 6
```

2차원 배열 선언 초기화

- **2차원 배열 선언 및 초기값 지정 방법 2가지**
  - 중괄호를 중첩되게 사용하는 방법: 권장 방법
  - 1차원 배열처럼 하나의 중괄호를 사용하는 방법
    - ➢ 권장하지 않음

권장 스타일

2차원 배열 형태로 초기화
- 행과 열의 수를 파악하기 쉬움

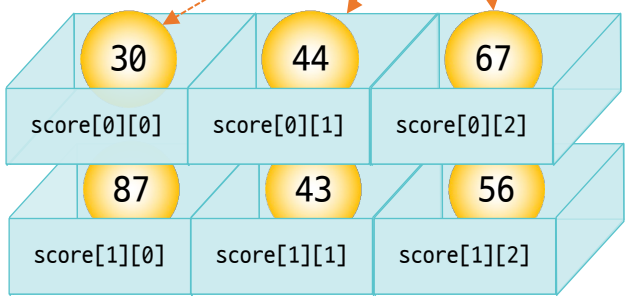
```
int score[2][3] = {
    {10, 20, 30},
    {40, 50, 60}};
```

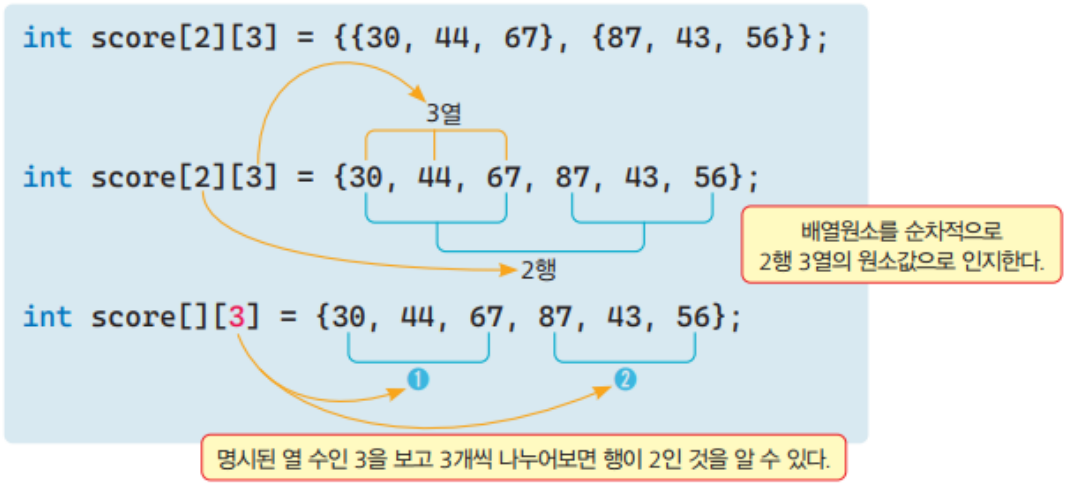
- **2차원 배열 초기값 지정**
  - 첫 번째 대괄호 내부의 행(row)의 크기는 생략 가능
    - ➢ 입력된 배열 원소 수와 열의 크기를 이용하여 행의 크기를 자동 계산
  - 두 번째 대괄호 내부의 열(column)의 크기는 반드시 입력해야 됨

행의 크기 생략 가능

```
int score[][3] = {
    {10, 20, 30},
    {40, 50, 60}};
```

```
int score[2][3] = {{30, 44, 67}, {87, 43, 56}};
```





2차원 배열 선언에서의 주의점

- **2차원 배열에서도 초기값이 총 배열 원소 수보다 적게 주어지면**
  - 나머지는 모두 기본값인 0, 0.0 또 는 '\0'이 저장

- **2차원 배열 선언 오류**
  - 반드시 열(column)의 크기는 입력해야 됨

```
int data[2][2] = {1, 2, 3, 4, 5};  // 원소 개수 초과
int data[2][2] = {{1, 2} {3, 4}};  // 쉼표(,) 없음
int data[2][] = {1, 2, 3, 4};      // 행의 크기만 입력
int data[][] = {1, 2, 3, 4};       // 행, 열 크기 없음
```

오류 수정

```
int data[2][2] = {1, 2, 3, 4};
int data[2][2] = {{1, 2}, {3, 4}};
int data[2][2] = {1, 2, 3, 4};
int data[][2] = {1, 2, 3, 4};
```

2차원 배열 초기화와 원소 출력

<04initarray.c>

```c
#include <stdio.h>

#define ROWSIZE 2
#define COLSIZE 3

int main(void)
{
    // 2차원 배열 초기화
    int td[][3] = { {1}, { 1, 2, 3 } };

    printf("반목문 for를 이용하여 출력\n");
    for (int i = 0; i < ROWSIZE; i++)
    {
        for (int j = 0; j < COLSIZE; j++)
            printf("%d ", td[i][j]);
        printf("\n");
    }

    return 0;
}
```

크기(2행 x 3열)는 자동 계산
- 입력되지 않은 값은 0으로 초기화

모든 배열의 원소가 0으로 초기화
- 일반적인 사용 방법
- 배열의 크기를 모두 정의

```c
int td[2][3] = {0};
```

실행 결과

```
반목문 for를 이용하여 출력
0 0 0
0 0 0
```

실행 결과

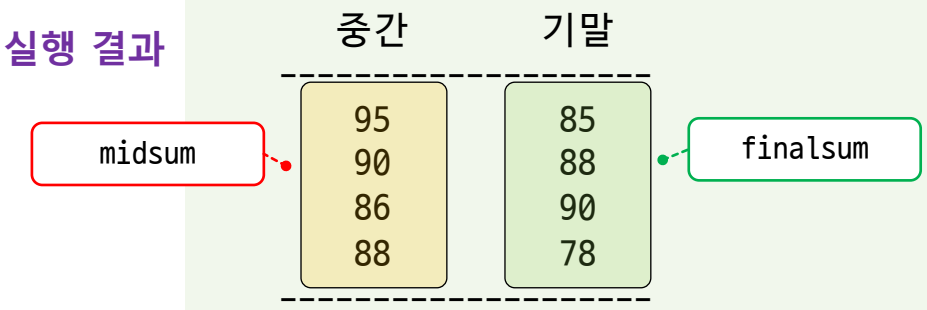
```
반목문 for를 이용하여 출력
1 0 0
1 2 3
```

2차원 배열 초기화를 이용한 성적 처리 (실습)

<05tdscore.c>

```c
#include <stdio.h>
#define ROWSIZE 4
#define COLSIZE 2

int main(void)
{
    int sum = 0, midsum = 0, finalsum = 0;

    //2차원 배열 초기화
    //int score[][COLSIZE] = { 95, 85, 90, 88, 86, 90, 88, 78 };
    int score[][COLSIZE] = {{95, 85},
                            {90, 88},
                            {86, 90},
                            {88, 78}};

    printf(" 중간 기말\n");
    printf(" -------------------\n");
    for (int i = 0; i < ROWSIZE; i++)
    {
        for (int j = 0; j < COLSIZE; j++)
        {
            printf("%10d ", score[i][j]);
            sum += score[i][j];
            if (j == 0)
                midsum += score[i][j];
            else
                finalsum += score[i][j];
        }
        puts("");
    }

    printf(" -------------------\n");
    printf("평균: %6.2f %10.2f\n", (double)midsum / ROWSIZE,
                                   (double)finalsum / ROWSIZE);
    printf("\n성적의 합은 %d이고 ", sum);
    printf("평균은 %.2f이다.\n",
                          (double)sum / (ROWSIZE * COLSIZE));

    return 0;
}
```

실행 결과




```
평균:  89.75      85.25

성적의 합은 700이고 평균은 87.50이다.
```

2차원 배열을 이용한 행렬 연산

<matrix.c>

```c
/**
 * 행렬 덧셈 연산
 */

#include <stdio.h>
#define ROWS 3
#define COLS 3

int main(void)
{
    int matrixA[ROWS][COLS] = {{2, 3, 0},
                               {8, 9, 1},
                               {7, 0, 5}};

    int matrixB[ROWS][COLS] = {{1, 0, 0},
                               {1, 0, 0},
                               {1, 0, 0}};

    int matrixC[ROWS][COLS];
    int r, c;

    // 두개의 행렬을 더한다.
    for (r = 0; r < ROWS; r++)
        for (c = 0; c < COLS; c++)
            matrixC[r][c] = matrixA[r][c] + matrixB[r][c];

    // 행렬을 출력한다.
    for (r = 0; r < ROWS; r++)
    {
        for (c = 0; c < COLS; c++)
            printf("%d ", matrixC[r][c]);
        printf("\n");
    }

    return 0;
}
```

실행 결과

```
3 3 0
9 9 1
8 0 5
```

$$\begin{pmatrix} 2 & 3 & 0 \\ 8 & 9 & 1 \\ 7 & 0 & 5 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 0 \\ 9 & 9 & 1 \\ 8 & 0 & 5 \end{pmatrix}$$

이미지 처리 예제

<imageproc.c>

```c
#include <stdio.h>
#define ROW 8
#define COL 16

int main(void)
{
    int image[ROW][COL] = {
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1},
        {1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
        {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
        {1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};

    printf("변환 후 이미지\n");
    for (int i = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++)
        {
            if(image[i][j] == 0)
                image[i][j] = 1;
            else
                image[i][j] = 0;

            printf("%d ", image[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

실행 결과

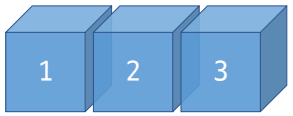
```
변환 후 이미지
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 1 1 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

3차원 배열 구조와 선언

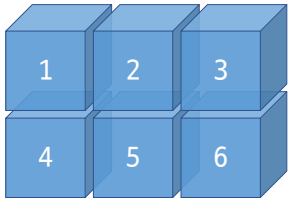
- 1차원, 2차원, 3차원 배열을 위한 선언과 그 구조
- 배열 선언
  - int threed[2][2][3];
    - 총 2×2×3 = 12 개의 배열 원소
    - 대괄호 내부 세 개의 크기는 모두 필요

```
int threed[2][2][3]; //총 2*2*3 = 12개 원소의 3차원 배열

threed[0][0][0] = 1; // 첫 번째 원소
threed[0][0][1] = 1; // 두 번째 원소
threed[0][0][2] = 1; // 세 번째 원소
threed[0][1][0] = 1; // 네 번째 원소
    ... (중간생략)
threed[1][1][2] = 1; // 열두 번째(마지막) 원소
```

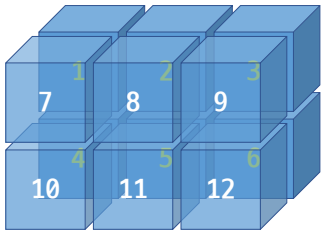
int a[3]
- 1차원 배열



int b[2][3]
- 2차원 배열

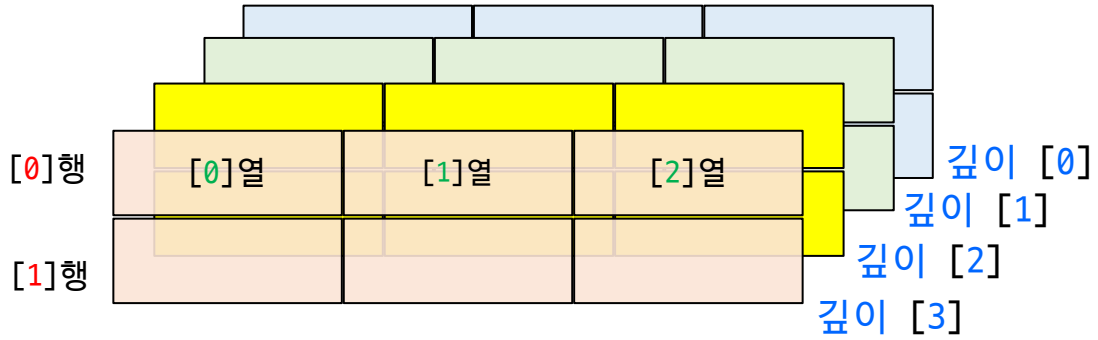


int c[2][2][3]
- 3차원 배열

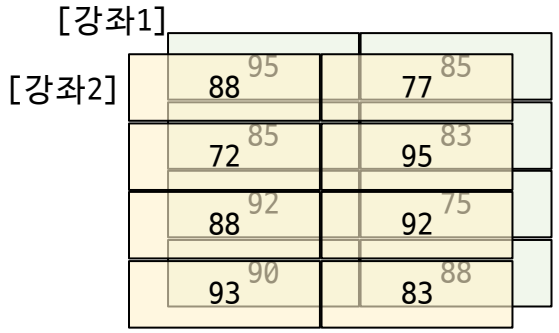


3차원 배열 선언

- **3차원 배열 선언**
  - 자료형 배열이름[depth][row][col]
  - int score[4][2][3]



- **3차원 배열 선언 및 초기화**
  - 2개의 강좌에 대한 학생들의 중간, 기말고사 성적 계산
  - 선언: int score[2][4][2];
    - depth 2: 강좌 수
    - row 4: 학생 수
    - col 2: 시험 회수(중간, 기말)



[강좌 1]	중간	기말
학생 1	95	85
학생 2	85	83
학생 3	92	75
학생 4	90	88

[강좌 2]	중간	기말
학생 1	88	77
학생 2	72	95
학생 3	88	92
학생 4	93	83

```
int score[2][4][2] = {
    { { 95, 85 },
      { 85, 83 },
      { 92, 75 },
      { 90, 88 } },
    { { 88, 77 },
      { 72, 95 },
      { 88, 92 },
      { 93, 83 } }
};
```

3차원 배열 초기화를 이용한 성적 점수 출력

<06thdary.c>

```c
#include <stdio.h>
#define ROWSIZE 4
#define COLSIZE 2

int main(void)
{
    // 3차원 배열 초기화, 첫 번째 크기는 지정하지 않을 수 있음
    int score[][ROWSIZE][COLSIZE] =
    {
                { { 95, 85 },
                  { 85, 83 },
                  { 92, 75 },
                  { 90, 88 } },
                { { 88, 77 },
                  { 72, 95 },
                  { 88, 92 },
                  { 93, 83 } }
    };
```

```c
    for (int i = 0; i < 2; i++)
    {
        if (i == 0)
            printf("[강좌 1]");
        else
            printf("[강좌 2]");
        printf("%11s%9s\n", "중간", "기말");

        for (int j = 0; j < ROWSIZE; j++)
        {
            printf("%10s%2d", "학생", j + 1);
            for (int k = 0; k < COLSIZE; k++)
                printf("%6d ", score[i][j][k]);
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

깊이(depth)

행(row)

열(column)

```
[강좌 1]     중간   기말
    학생 1    95     85
    학생 2    85     83
    학생 3    92     75
    학생 4    90     88

[강좌 2]     중간   기말
    학생 1    88     77
    학생 2    72     95
    학생 3    88     92
    학생 4    93     83
```

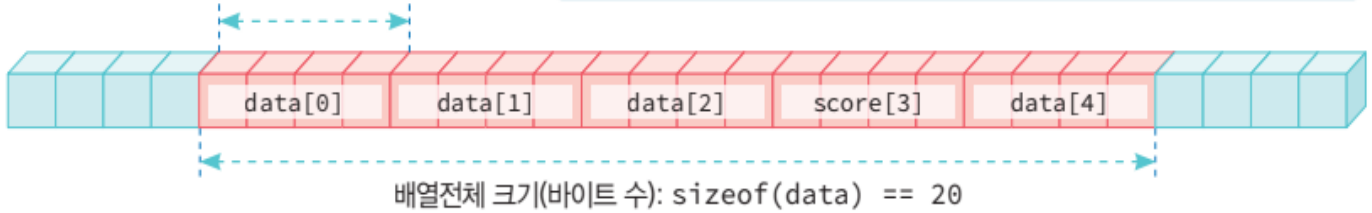
배열 크기 연산

- **sizeof() 연산 사용**
  - 저장 공간의 크기를 바이트 수로 반환하는 연산자

- **배열의 크기 계산**
  - sizeof(배열이름) / sizeof(배열원소) = 20 bytes / 4 bytes = 5개
    - sizeof(배열이름): 배열의 전체 공간의 바이트 수
    - sizeof(배열원소): 배열 원소 하나의 바이트 수

```
int data[] = {12, 23, 17, 32, 55};
```

```
배열크기(배열원소의 수) = sizeof(배열이름) / sizeof(배열원소)

int  arraysize = sizeof(data) / sizeof(data[0]);
```

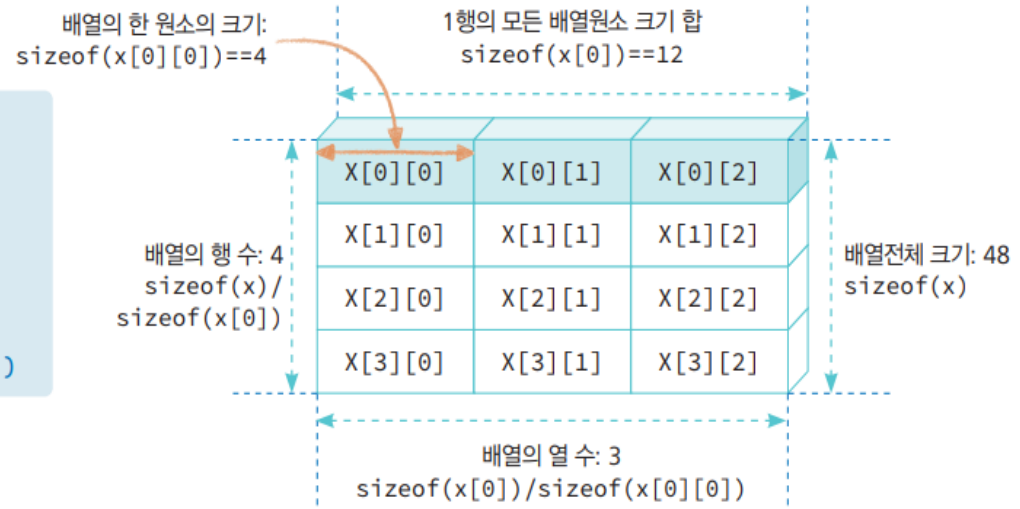


2차원 배열 크기 계산방법

- **2차원 배열의 행(row)의 수**
  - `sizeof(x) / sizeof(x[0])`
    - sizeof(배열이름): 배열의 전체 공간의 바이트 수
    - sizeof(x[0]): 첫 행의 바이트 수

- **2차원 열(column)의 수**
  - `( sizeof(x[0]) / sizeof(x[0][0]) )`
    - sizeof(x[0][0]): 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])
```



1차원과 2차원 배열에서 배열 전체 및 원소의 크기

<07arysize.c>

```c
#include <stdio.h>

int main(void)
{
    int data[] = { 3, 4, 5, 7, 9 };

    printf("%zd %d\n", sizeof(data), (int) sizeof(data[0]));
    printf("배열 data 크기 == %d\n", (int) (sizeof(data) / sizeof(data[0])));

    //4 x 3 행렬
    double x[][2] = {
        { 1.2, 2.3},
        { 7.3, 8.9}};

    printf("%d %d ", (int) sizeof(x), (int) sizeof(x[0])); //전체 크기와 첫 행의 크기
    printf("%d %d\n", (int) sizeof(x[1]), (int) sizeof(x[0][0]));

    int rowsize = sizeof(x) / sizeof(x[0]);
    int colsize = sizeof(x[0]) / sizeof(x[0][0]);
    printf("이차원 배열 x: 행수 = %d 열수 = %d\n", rowsize, colsize);
    printf("이차원 배열 x: 전체 원소 수 = %d\n", (int) (sizeof(x) / sizeof(x[0][0])));

    return 0;
}
```

```
20 4
배열 data 크기 == 5
32 16 16 8
이차원 배열 x: 행수 = 2  열수 = 2
이차원 배열 x: 전체 원소 수 = 4
```

LAB 2차원 배열에서 원소 참조 방법과 저장 값을 출력

<lab2aryprint.c>

```c
#include <stdio.h>

int main()
{
    int a[3][4] =
    {
        { 1, 2, 7, 3 }, /* initializers for row indexed by 0 */
        { 5, 6, 3, 4 }, /* initializers for row indexed by 1 */
        { 9, 7, 1, 8 } /* initializers for row indexed by 2 */
    };

    printf(" %7s %5s ", "원소", "값");
    printf(" %7s %5s ", "원소", "값");
    printf(" %7s %5s ", "원소", "값");
    printf(" %7s %5s\n", "원소", "값");
    printf("-----------------------------------------------\n");

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 4; j++)
            printf(" a[%d][%d]: %d ", i, j, a[i][j]);
        puts("");
    }
    return 0;
}
```

```
  원소    값     원소    값     원소    값     원소    값
-----------------------------------------------
 a[0][0]: 1   a[0][1]: 2   a[0][2]: 7   a[0][3]: 3
 a[1][0]: 5   a[1][1]: 6   a[1][2]: 3   a[1][3]: 4
 a[2][0]: 9   a[2][1]: 7   a[2][2]: 1   a[2][3]: 8
```

Questions?