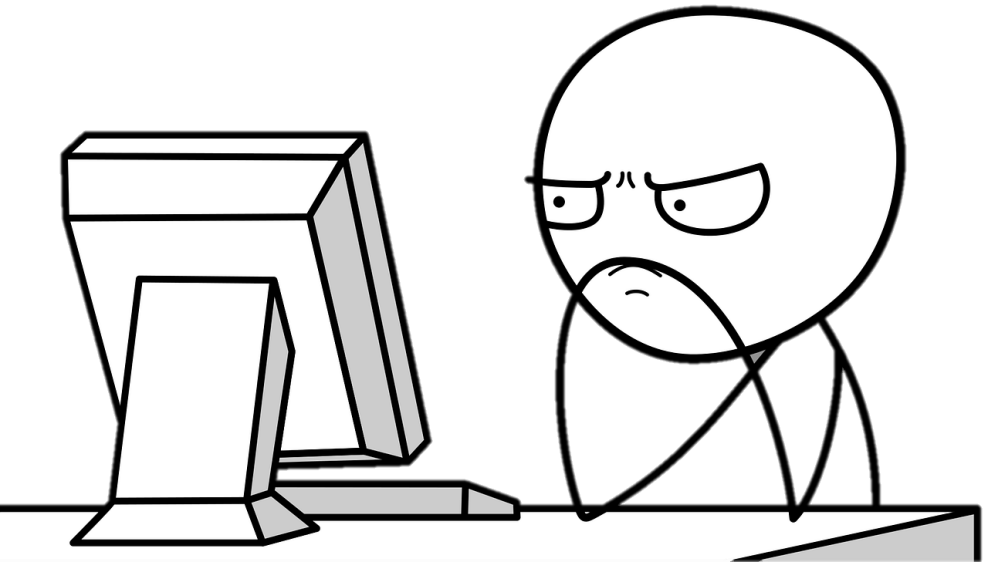

컴퓨터 프로그래밍
Computer Programming


09
배열

목차

1. 배열 선언과 초기화



배열array의 필요성

- 40명 학생 성적을 변수에 저장하려면 40개의 변수를 선언
- 여러 변수들이 같은 배열이름으로 일정한 크기의 연속된 메모리에 저장되는 구조가 있다면 편리
- 변수를 일일이 선언하는 번거로움을 해소
- 배열을 구성하는 각각의 변수를 참조하는 방법도 간편

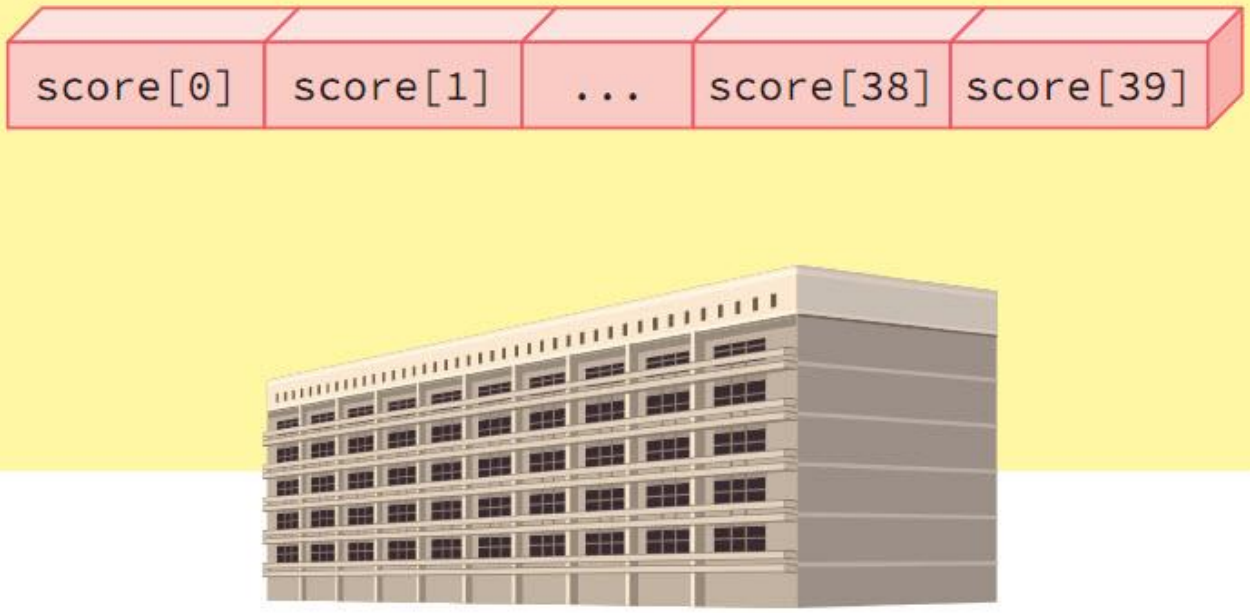
배열 정의

- 한 자료유형의 저장공간인 원소를 동일한 크기로 지정된 배열크기만큼 확보한 연속된 저장공간
- 배열을 구성하는 각각의 항목을 배열의 원소elements
- 배열원소는 첨자index 번호라는 숫자를 이용해 쉽게 접근
- 배열이름, 원소 자료유형, 배열크기

배열의 필요성과 개념

일반 변수의 활용

여러 값을 저장하기 위한 배열의 활용



배열선언 구문

- 원소자료유형 배열이름[배열크기];
  - int data[10];
  - 배열크기
    - 배열선언 시 초기값 지정이 없다면 반드시 배열크기는 양의 정수로 명시
    - 대괄호(bracket) 사이에 [배열크기]와 같이 기술
    - 양수 정수로 리터럴 상수와 매크로 상수 또는 이들의 연산식이 가능
    - 변수와 const 상수로는 배열의 크기를 지정 불가능

배열선언 구문

배열선언

원소자료형 배열이름[배열크기];

배열크기는 리터럴 상수, 매크로 상수
또는 이들의 연산식이 허용된다

```
#define SIZE 5

int score[10];
double point[20];
char ch[80];
float grade[SIZE];
int score[SIZE+1];
int degree[SIZE*2];
```

오류발생

```
int n = 5;

int score[n];
double point[-3];
char ch[0];
float grade[3.2];
int score[n+2];
int degree[n*2];
```

잘못된 배열 선언

```
int n = 5;

int score[n];
int grade[3 + 4];
```

잘못된 배열 선언

```
const int size = 6;

int cpoint[size];
double width[size + 4];
```

배열원소 접근

- 배열선언 후 배열원소에 접근
  - 배열이름 뒤에 대괄호 사이 첨자index를 이용
    - 첫 번째 배열원소를 접근하는 첨자값은 0, 다음 두 번째 원소는 1
    - 그 다음 원소를 접근하려면 순차적으로 1씩 증가
    - 배열에서 유효한 첨자의 범위는 0부터 (배열크기-1)까지
    - 첨자의 유효 범위를 벗어나 원소를 참조하면 문법오류 없이 실행오류가 발생
  - 배열선언 시 대괄호 안의 수는 배열 크기
    - 선언 이후 대괄호 안의 수는 원소를 참조하는 번호인 첨자
- 초기값
  - 배열을 함수내부에서 선언 후 원소에 초기값을 저장하지 않으면
  - 쓰레기값이 저장되니 항상 초기값을 저장

배열선언과 원소참조

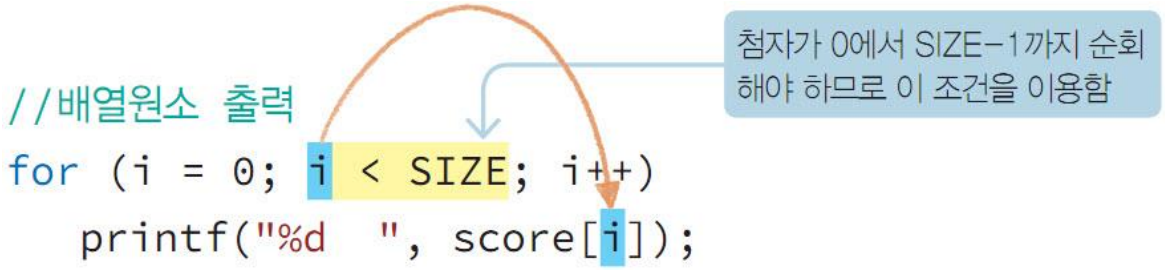
```
int score[5];

//배열 원소에 값 저장
score[0] = 78;
score[1] = 97;
score[2] = 85;
//배열 4번째 원소에 값 저장하지 않아 쓰레기값 저장
score[4] = 91;
score[5] = 50; //문법오류는 발생하지 않으나 실행오류 발생
```

Source Code #01: declarearray.c

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

#define SIZE 5

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

    //배열 원소에 값 저장
    score[0] = 78;
    score[1] = 97;
    score[2] = 85;
    //배열 4번째 원소에 값 저장하지 않아 쓰레기 값 저장
    score[4] = 91;
    //score[5] = 50; //문법오류는 발생하지 않으나 실행오류 발생

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

    return 0;
}
```

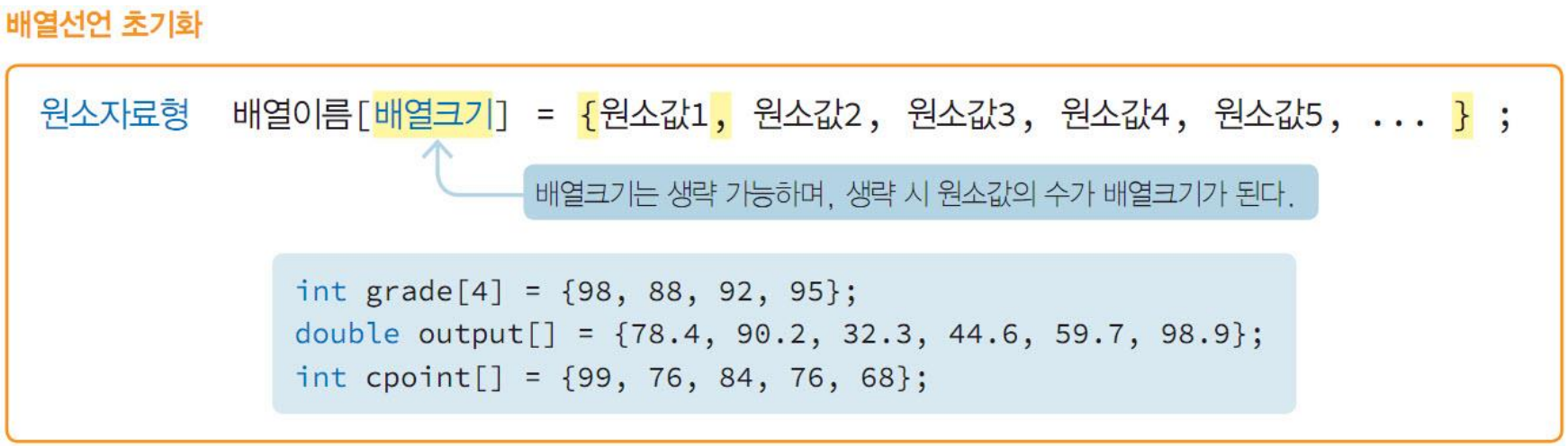
```
78  97  85  -858993460  91
```

- 배열 선언과 모든 원소 출력
- 반복 for 문의 제어변수를 0에서 시작하여 배열크기보다 작을 때까지 출력을 반복
  - 배열원소 score[3]에는 초기값을 저장하지 않고 출력하면 쓰레기값이 출력되는 것을 확인
  - 만일 20행의 주석을 제거하면 배열첨자의 유효범위를 초과한 5를 참조하므로 실행오류가 발생

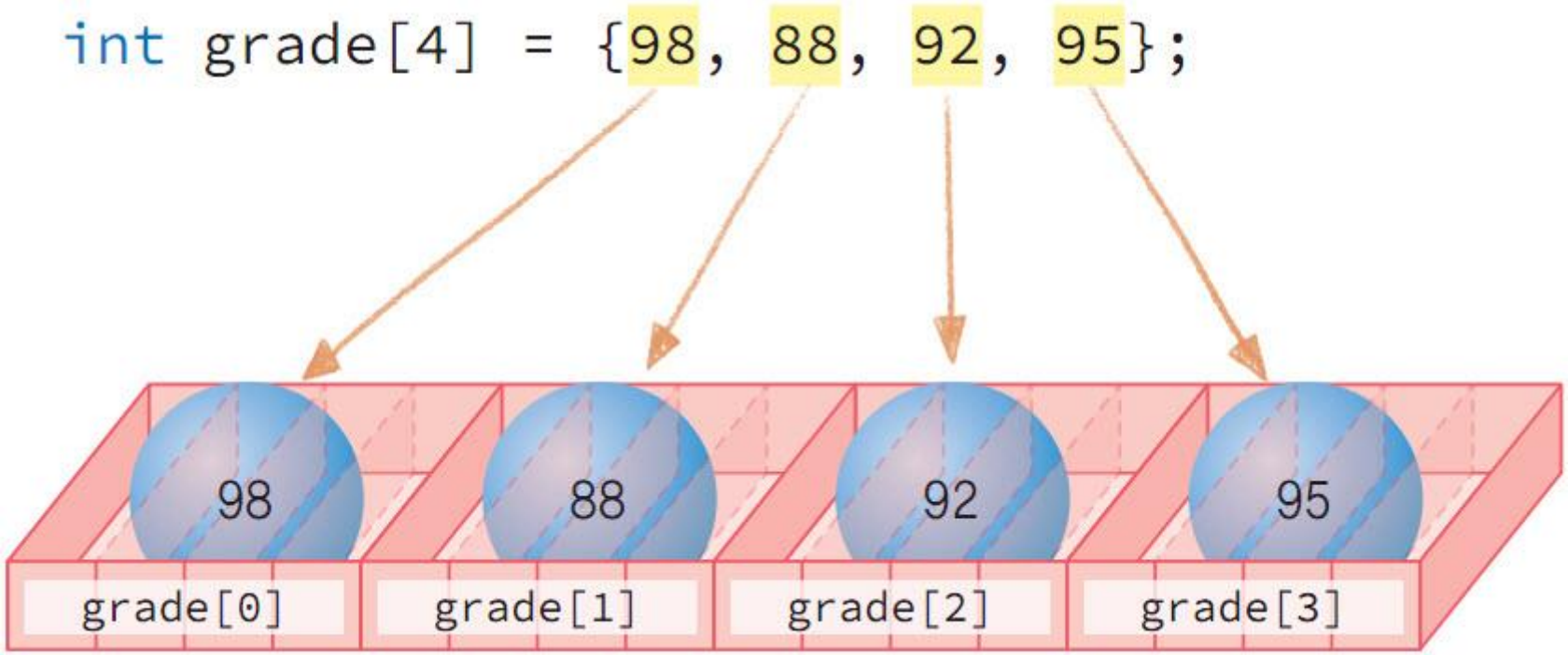


배열선언 초기화

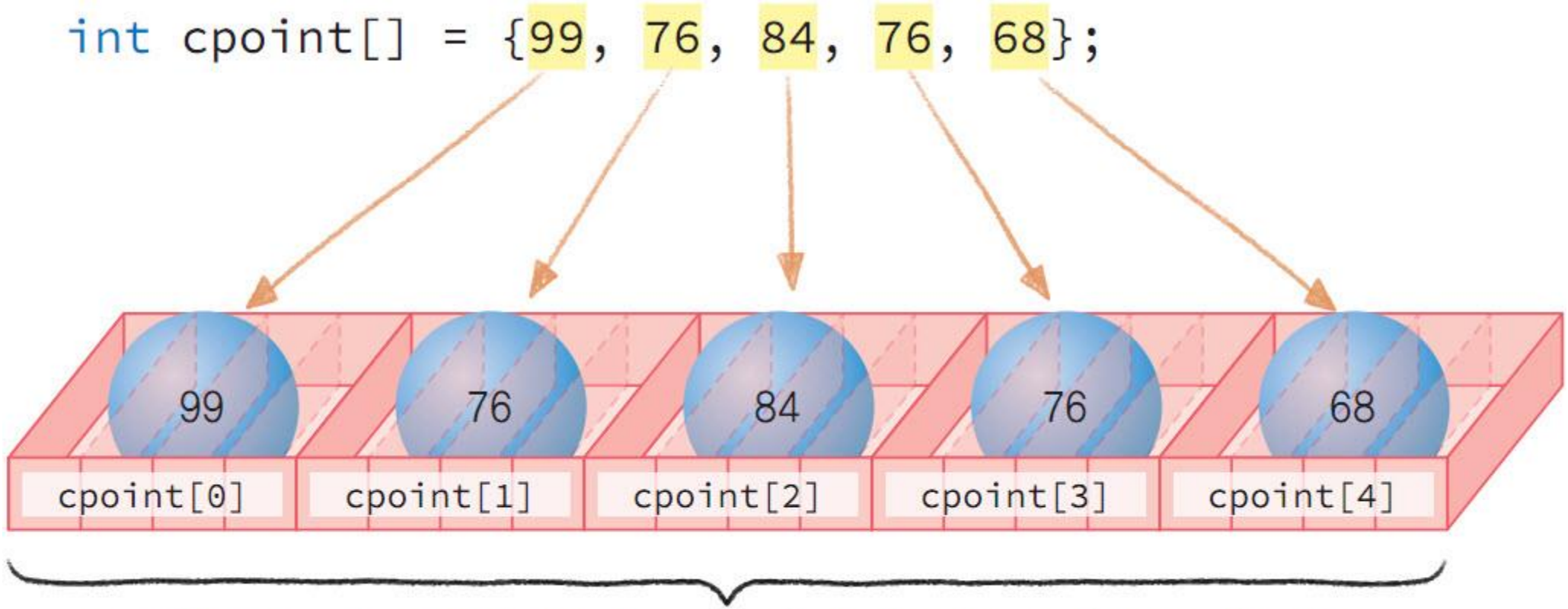
- 초기화 방법
  - 배열선언을 하면서 대입연산자를 이용
  - 중괄호 사이에 여러 원소값을 쉼표로 구분하여 기술
  - 배열크기를 넘지 않게 원소값을 나열
- 배열크기는 생략 가능
  - 생략하면 자동으로 중괄호 사이에 기술된 원소 수가 배열크기



배열선언 초기화



배열선언 초기화



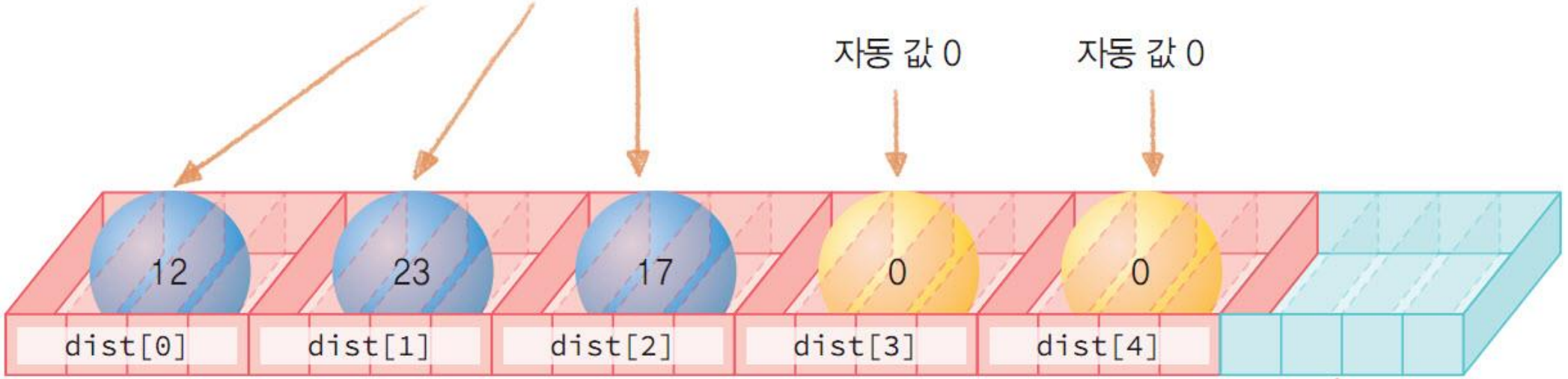
5: 배열크기를 지정하지 않으면 자동으로 초기값 지정 원소 수가 배열크기가 된다.

배열 초기화에서 배열크기

- 배열크기가 초기값 원소 수보다 크면
  - 지정하지 않은 원소의 초기값은 자동으로 모두 기본값으로 저장
  - 기본값이란 자료형에 맞는 0을 저장
  - 즉 정수형은 0, 실수형은 0.0 그리고 문자형은 '\0'인 널문자
- 반대로 배열크기가 초기값 원소 수보다 작으면
  - 배열 저장공간을 벗어나므로 "이니셜라이저가 너무 많습니다."라는 문법오류가 발생

배열 초기화에서 배열크기

```
int dist[5] = {12, 23, 17};
```



배열크기를 지정한 후 초기값 지정 원소 수가
배열크기보다 많으면 다음의 문법오류가 발생한다.

```
int dist[5] = {12, 23, 17, 55, 57, 71};
```

error C2078: 이니셜라이저가 너무 많습니다.

```
int dist[5] = {0};
```

지정한 배열크기보다 초기값 수가 적으면 모두 0으로
채워지므로 모든 배열 원소가 0으로 채워진다.

Source Code #02: initarray.c

```c
// file: initarray.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;
}
```

- 배열 score를 선언하면서 초기값으로 6개의 점수를 저장한 후 합과 평균을 구하여 출력

```
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.
```

초기화 주의

- 중괄호를 사용한 초기화 방법은 반드시 배열선언 시에만 이용이 가능하며 배열선언 이후에는 사용 불가능

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};
```

잘못된 초기화 문장

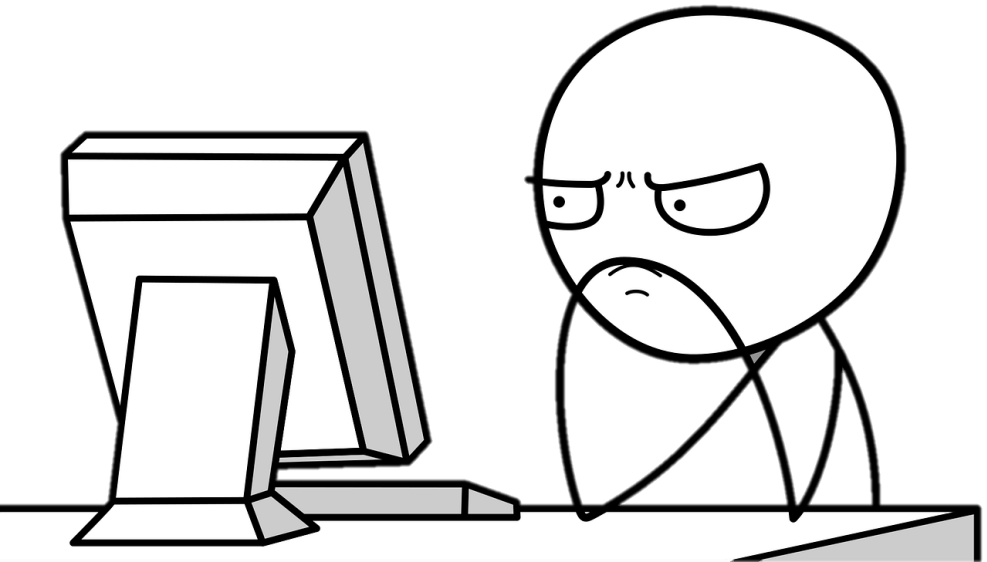
```
int n = 5;

int score[n] = {89, 92, 91};
int grade[3] = {98, 88, 92, 95};
int cpoint[] = {99  76  84  76  68  93};
char ch[] = {a, b, c};
double width[4]; width = {23.5, 32.1};
```

Lab #01: int 형 배열에 표준입력으로 받은 정수를 저장하여 출력

- 자료형 int로 선언된 배열 input에서 표준입력으로 받은 정수를 순서대로 저장하여 출력
  - 배열 input을 선언하면서 초기화로 모두 0을 저장
  - 표준입력으로 받은 정수는 0이 입력될 때까지 저장
  - 입력된 배열은 while 문을 사용하여 마지막 0 값 이전까지 다음 결과와 같이 출력
    - 배열에 저장할 정수를 여러 개 입력하시오. 0을 입력하면 입력을 종료합니다.
    - 30 26 65 39 87 76 0
    - 30 26 65 39 87 76

```c
// file: inputarray.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

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

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

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

    return 0;
}
```

2. 이차원과 삼차원 배열



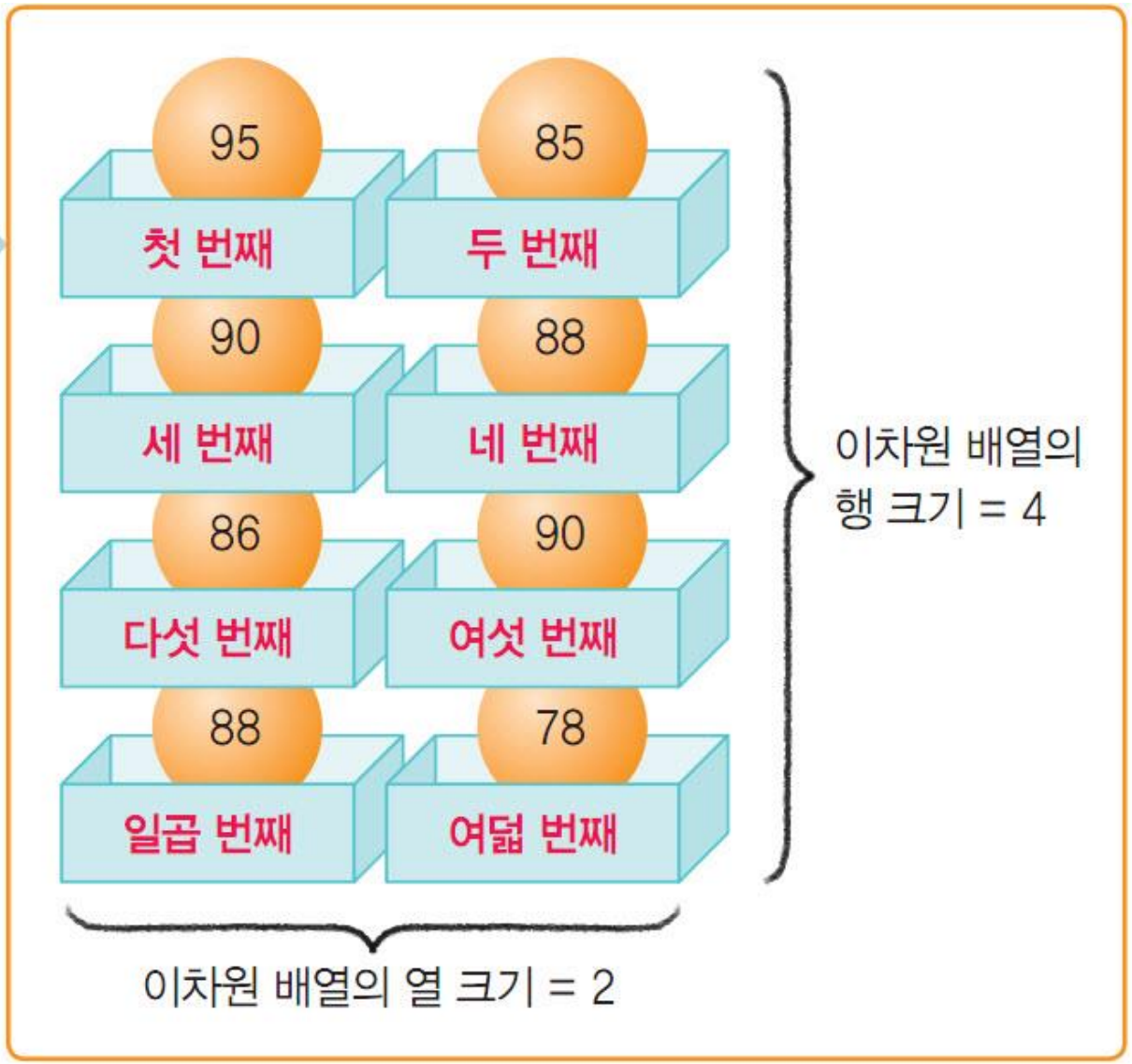
이차원 배열 개요

- 이차원 배열은 테이블 형태의 구조
- 4행 2열(4 x 2)의 이차원 배열 필요
- 총 배열원소는 8개

이차열 배열의 구조

4행 2열의 이차원 배열로 총 8개의 배열원소가 있는 구조

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



이차원 배열선언

- 이차원 배열선언은 2개의 대괄호가 필요
  - 첫 번째 대괄호에는 배열의 행 크기
  - 두 번째는 배열의 열 크기를 지정
  - 배열선언 시 초기값을 저장하지 않으면
    - 반드시 행과 열의 크기는 반드시 명시

이차원 배열선언

원소자료형 배열이름[배열행크기][배열열크기];

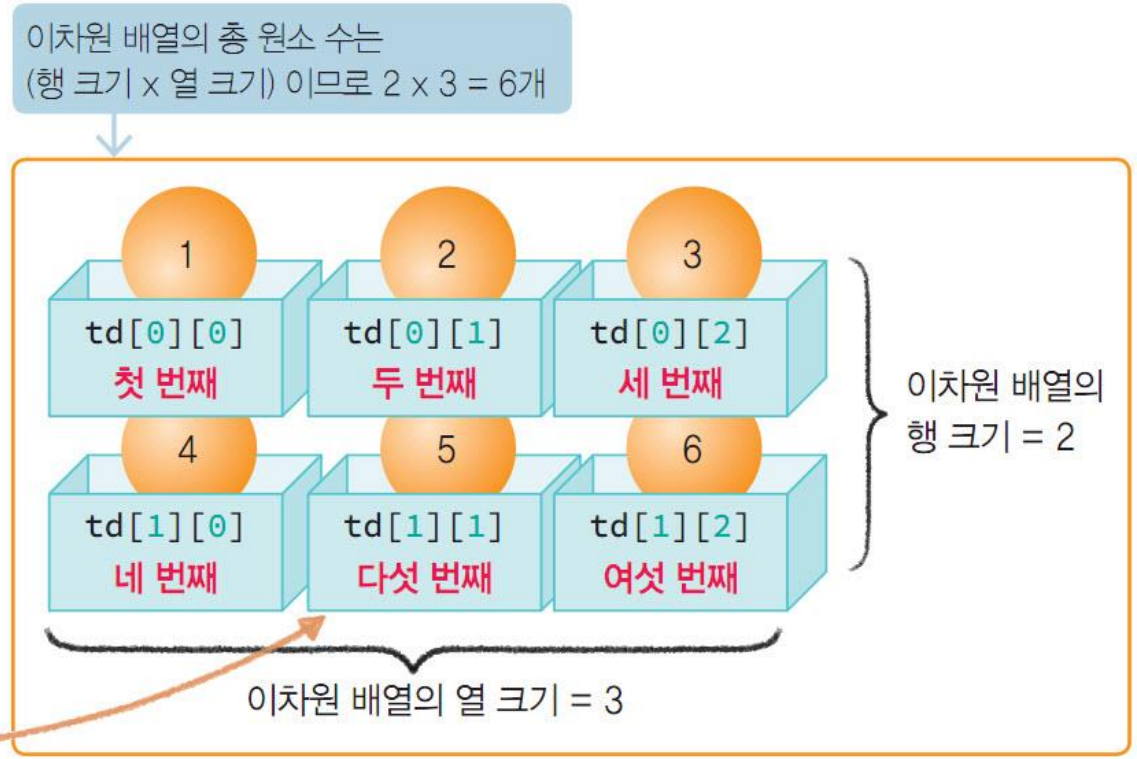
배열선언 시 배열크기는 생략할 수 없으며
배열크기는 리터럴 상수, 매크로 상수
또는 그들의 연산식이 허용된다.

```
#define RSIZE 5
#define CSIZE 2

int score[RSIZE][CSIZE];

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

이차원 배열 구조

- 이차원 배열원소를 참조
  - 행 첨자는 0에서 (행크기-1)까지 유효
  - 열 첨자는 0에서 (열크기-1)까지 유효

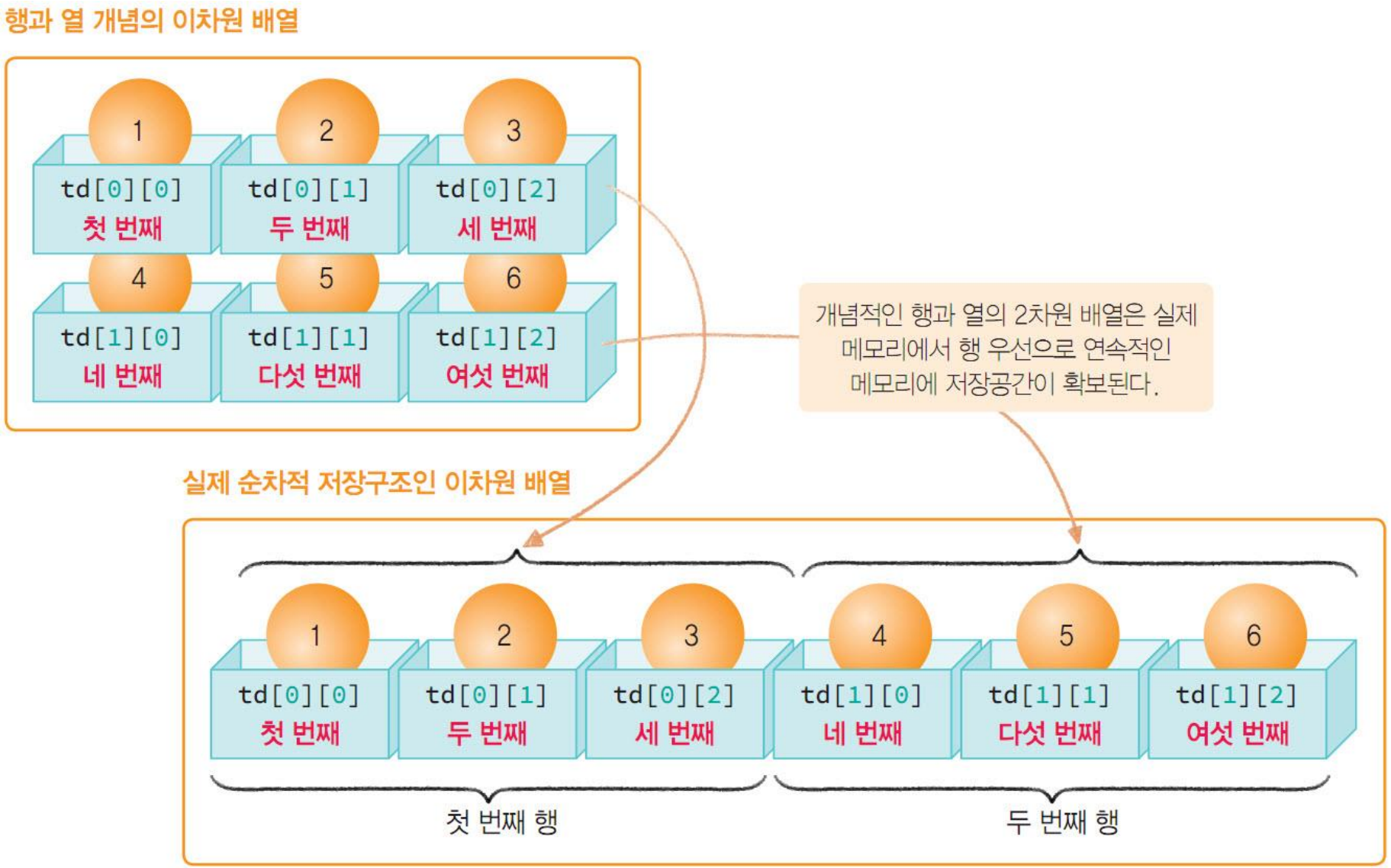
```
#define ROWSIZE 2
#define COLSIZE 3

// 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;
```



이차원 배열 구조

- 실제로 이차원 배열이 메모리에 저장될 때는 행과 열의 개념이 아니라 일차원과 같은 연속적인 메모리 공간에 저장
- 첫 번째 행 모든 원소가 메모리에 할당된 이후에 두 번째 행의 원소가 순차적으로 할당

이차원 배열 구조


행과 열 개념의 이차원 배열
1
2
3
td[0][0]
첫 번째
td[0][1]
두 번째
td[0][2]
세 번째
4
5
6
td[1][0]
네 번째
td[1][1]
다섯 번째
td[1][2]
여섯 번째
개념적인 행과 열의 2차원 배열은 실제 메모리에서 행 우선으로 연속적인 메모리에 저장공간이 확보된다.
실제 순차적 저장구조인 이차원 배열
1
2
3
4
5
6
td[0][0]
첫 번째
td[0][1]
두 번째
td[0][2]
세 번째
td[1][0]
네 번째
td[1][1]
다섯 번째
td[1][2]
여섯 번째
첫 번째 행
두 번째 행

Source Code #03: twodarray.c

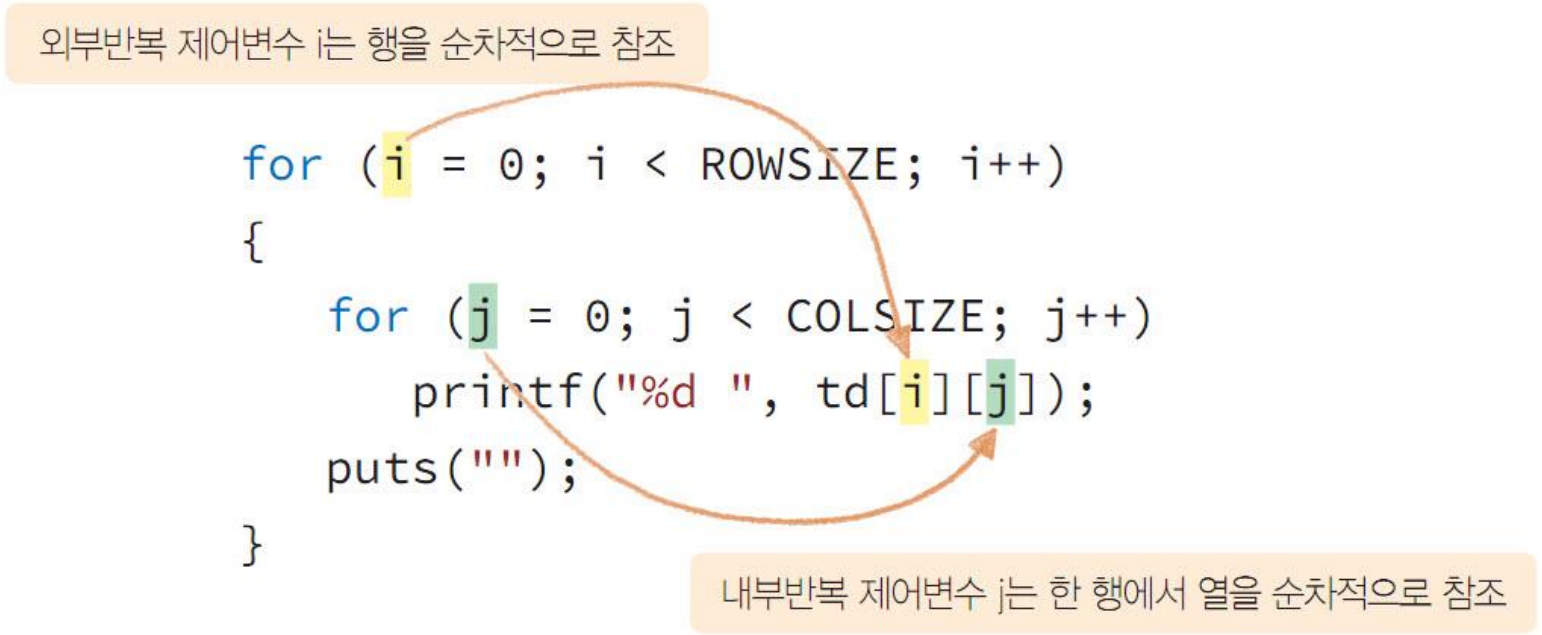
```c
// file: twodarray.c
#include <stdio.h>

#define ROWSIZE 2
#define COLSIZE 3

int main(void)
{
    // 2차원 배열선언
    int td[ROWSIZE][COLSIZE];

    // 2차원 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");
    }

    return 0;
}
```

- 이차원 배열 저장과 출력

```
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
```

이차원 배열 출력

- 외부반복: 제어변수 i는 행을 0에서 (행의 수-1)까지 순차적으로 참조
- 내부반복: 제어변수 j는 0에서 (열의 수-1)까지 열



이차원 배열 출력

- 배열원소의 저장값이 행과 열의 관계식 가능하면 중복된 반복문을 사용하여 값을 저장

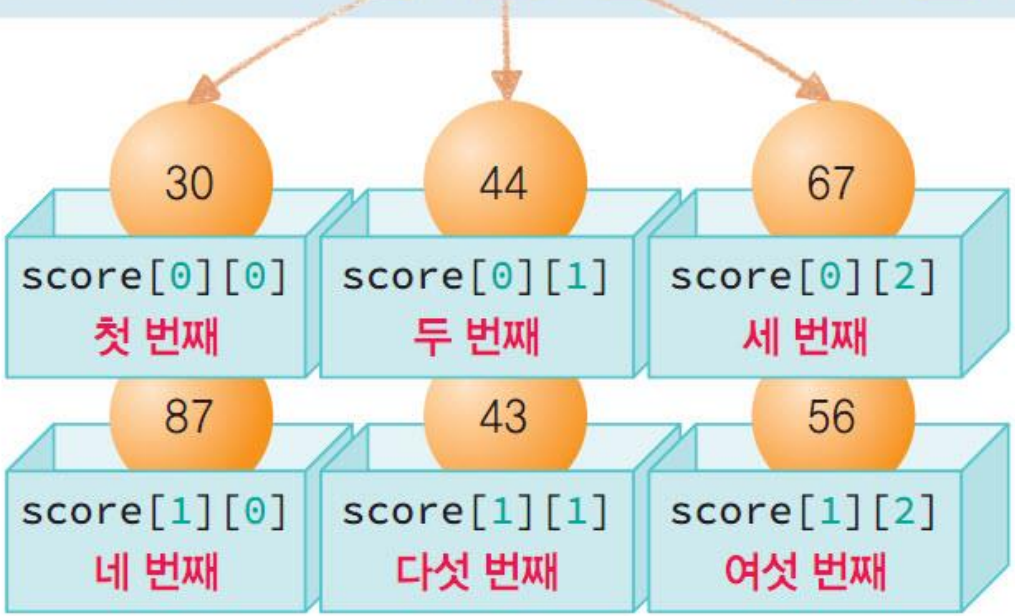
```
for (i = 0; i < ROWSIZE; i++)
    for (j = 0; j < COLSIZE; j++)
        td[i][j] = i*COLSIZE + j + 1;
```

```
// 2차원 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;
```

이차원 배열선언 초기화

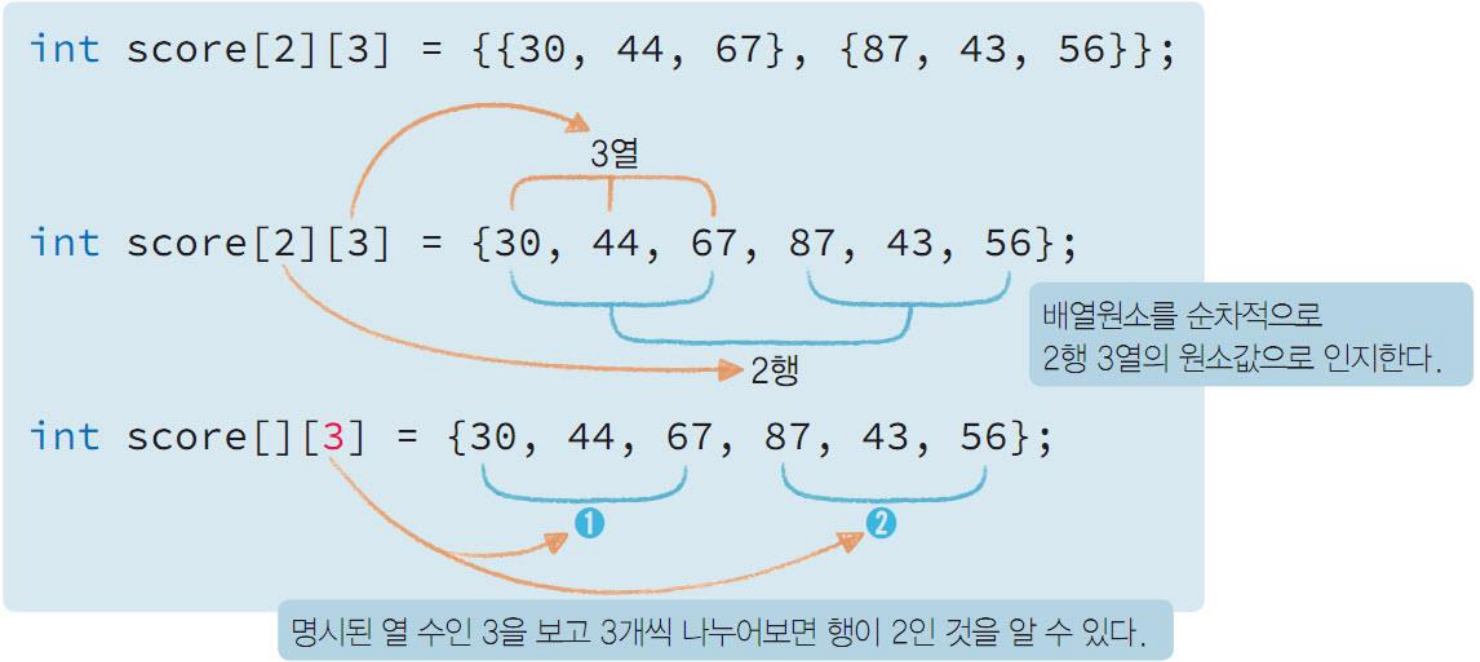
- 이차원 배열을 선언하면서 초기값을 지정하는 방법 2가지
  - 중괄호를 중첩되게 이용하는 방법
  - 하나의 중괄호를 사용하는 방법

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



이차원 배열선언 초기화

- 이차원 배열선언 초기값 지정에서도 첫 번째 대괄호 내부의 행의 크기는 명시하지 않을 수 있음
  - 두 번째 대괄호 내부의 열의 크기는 반드시 명시
  - 행 크기 없이 열 크기만 명시한다면
    - 명시된 배열원소 수와 열 크기를 이용하여 행의 크기를 자동으로 산정
    - ((배열원소 수) / (열 수)) 에서 소수점인 경우 무조건 올림 하면 행의 수



Source Code #04: inittwodarray.c

```c
// file: inittwodarray.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;
}
```

- 이차원 배열 초기화와 활용

```
반목문 for를 이용하여 출력
1 0 0
1 2 3
```

초기화 주의점

- 열 크기는 생략할 수 있어도 행 크기는 절대 생략할 수 없음

```
int data[2][2] = {1, 2, 3, 4, 5};  //원소 수 초과
int data[2][2] = {{1, 2}  {3, 4}}; //쉼표 , 빠짐
int data[2][] = {1, 2, 3, 4};      //행 크기만 기술
int data[][] = {1, 2, 3, 4}; //행, 열 크기 모두 없음
```

오류 수정

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

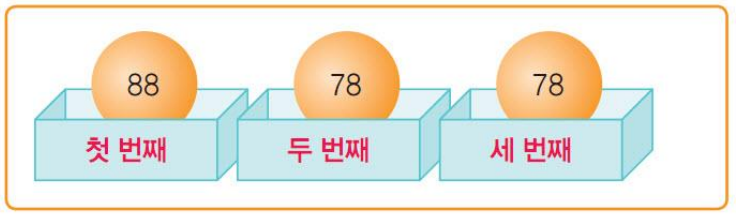
삼차원 배열

- 삼차원 배열 구조
  - 다차원 배열을 지원
- 배열 선언
  - 배열선언 시 대괄호 내부 3개의 크기는 모두 필요

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

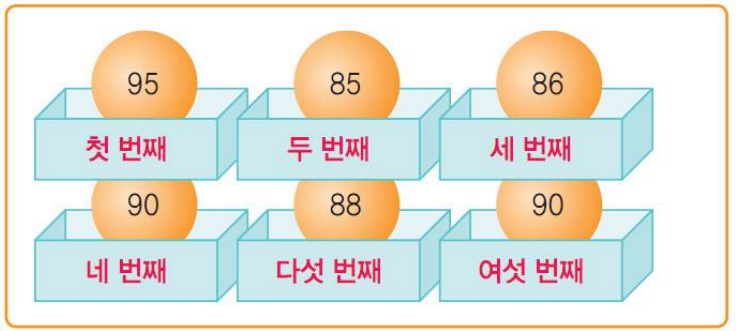
```
int a[3];
```

배열크기 3의 일차원 배열



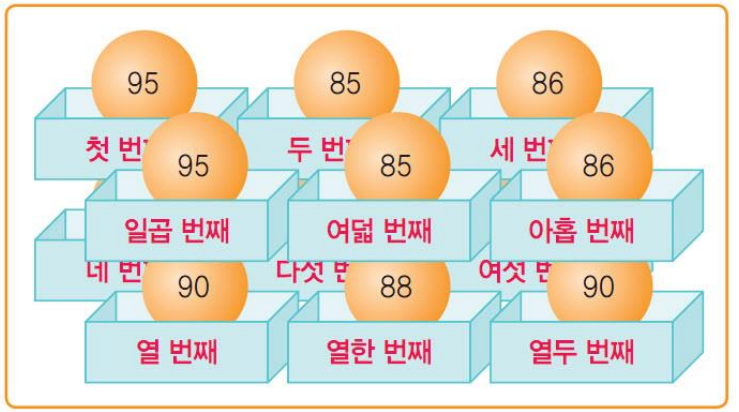

```
int b[2][3];
```

2행 3열의 이차원 배열




```
int c[2][2][3]
```

2 x 2 x 3의 삼차원 배열



Source Code #05: threedary.c

강좌 2개의 각반 4명에 대한 중간과 기말고사 성적을 초기화하여 출력하는 프로그램

```c
// file: threedary.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 } }
    };

    for (int i = 0; i < 2; i++)
    {
        if (i == 0) printf("[강좌 1]");
        else printf("[강좌 2]");
        printf("%11s%7s\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;
}
```

삼차원 배열 초기화

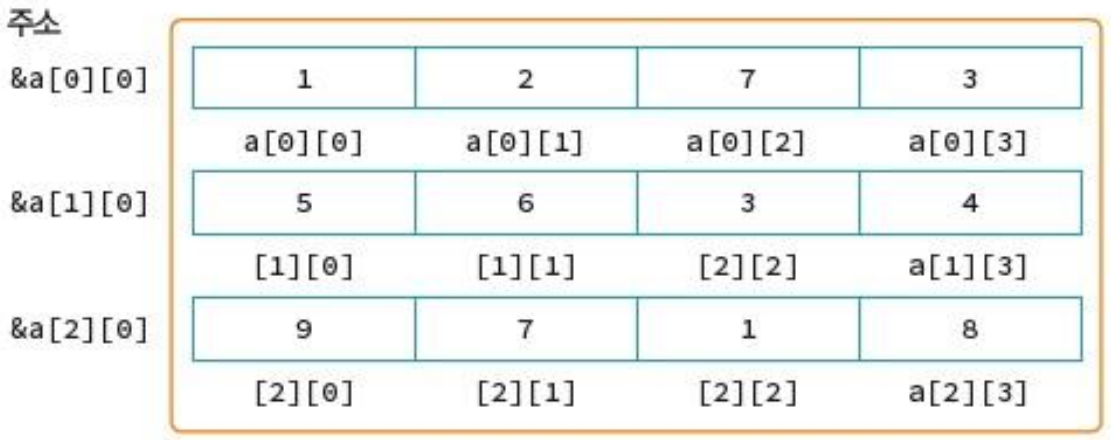
- 만일 강좌 2개에 대한 C 언어 성적 저장을 위한 배열
  - 3차원 배열 score[2][4][2]로 선언
  - 학생의 점수를 초기값으로 저장
- 순서대로 첫 번째 상수인 2는 강좌의 수
  - 두 번째 상수 4는 각 반의 학생 수
  - 마지막 세 번째 상수 2는 중간과 기말고사인 시험 횟수

```
[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 } }
};
```

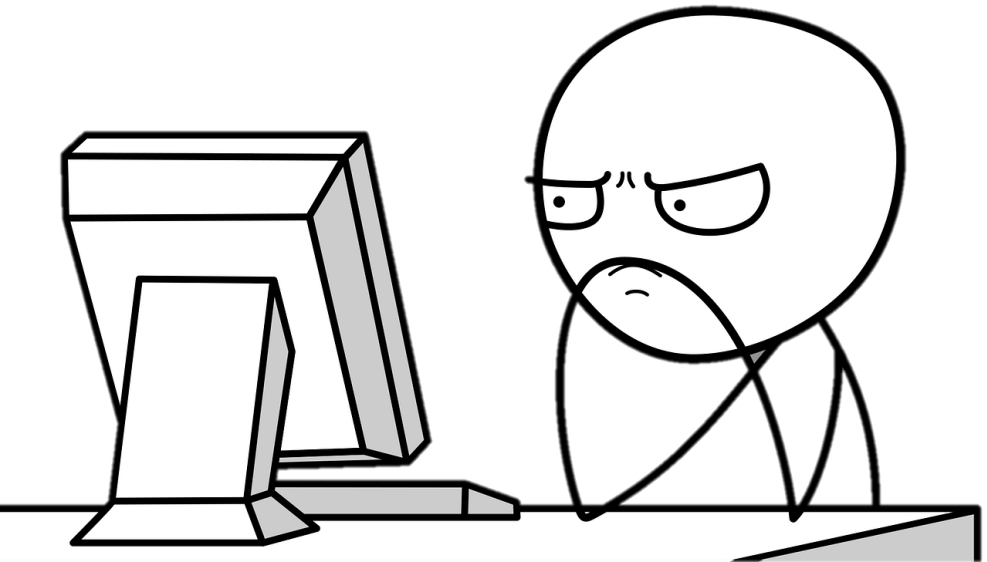
Lab #02: 이차원 배열에서 원소참조, 주소와 저장값을 출력

- 자료형 int로 선언된 이차원 배열 a에서 초기화로 값을 저장
  - 배열의 모든 원소를 순서대로 원소참조, 주소와 저장값을 출력하는 프로그램
  - 이차원 배열 구조와 저장값은 다음과 같으며 초기화는 원소를 콤마로 구분
  - 배열 원소의 주소와 저장값은 각각 &a[i][j]와 a[i][j]로 참조할 수 있다.




```c
// file: arrayprint.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("%6s %6s %3s  ", "원소", "주소", "값");
    printf("%6s %6s %3s  ", "원소", "주소", "값");
    printf("%6s %6s %3s  ", "원소", "주소", "값");
    printf("%6s %6s %3s\n", "원소", "주소", "값");
    printf("-----------------------------------");
    printf("-----------------------------------\n");

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

    return 0;
}
```

3. 배열과 포인터 관계

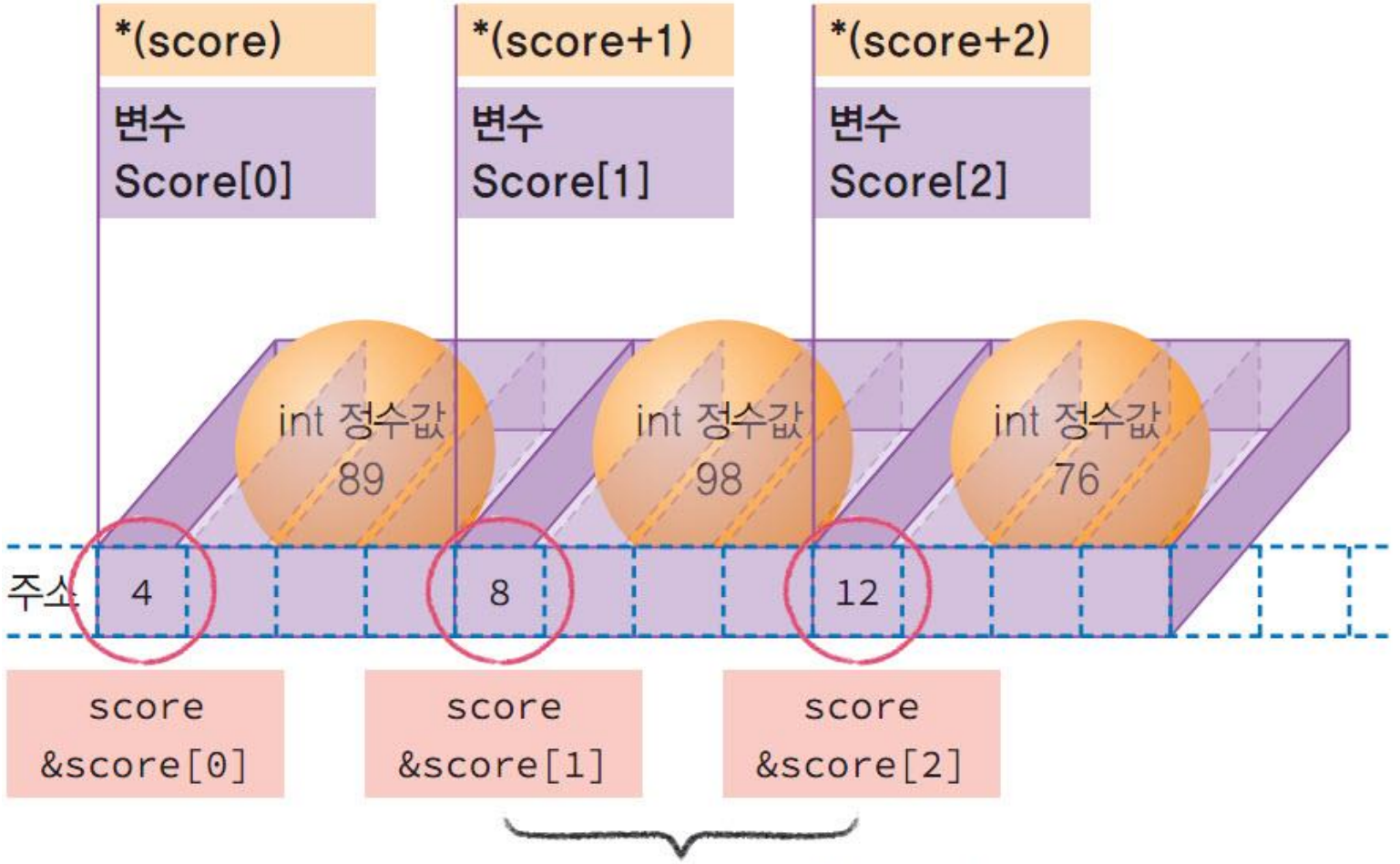


배열이름을 이용한 참조

- 배열 score에서 배열이름 score 자체가 배열 첫 원소의 주소값인 상수
  - score와 &score[0]는 같음
- 배열이름 score를 이용하여 모든 배열원소의 주소와 저장값을 참조 가능
  - 간접연산자를 이용한 *score는 변수 score[0]
- (score + 1)은 &score[1]
  - 이것을 확장하면 (score + i)는 &score[i]
- 다시 간접연산자를 사용하면
  - *(score + i)는 score[i]

배열이름을 이용한 참조

```
int score[] = {89, 98, 76};
```



score+1와 score+2의 차이는 원소크기인 4이다.

주소값과 참조값

- 주소값 (score + 1)을 출력
  - score 주소값에 int 형의 크기인 4를 더한 주소값
- 연산식 (score + i)
  - score 다음 i번째 원소의 주소값을 알아내는 연산식
- 간접연산자 *를 사용한 연산식 *(score + i)
  - 배열 score의 (i+1) 번째 배열원소로 score[i]
  - 연산식 *(score+1)과 (*score + 1)은 다르므로 주의
  - (*score + 1)은 배열 첫 번째 원소에 1을 더하는 연산식

배열 원소의 주소와 내용 값의 다양한 접근방법

배열 초기화 문장		int score[] = {89, 98, 76};		
배열 원소값		89	98	76
배열원소 접근방법	score[i]	score[0]	score[1]	score[2]
	*(score+i)	*score	*(score+1)	*(score+2)
주소값 접근 방법	&score[i]	&score[0]	&score[1]	&score[2]
	score+i	score	score+1	score+2
실제 주소값	base + 원소크기*i	만일 4라면	8 = 4+1*4	12 = 4+2*4

Source Code #06: array.c

```c
// file: array.c
#include <stdio.h>
#define SIZE 3

int main(void)
{
    int score[] = { 89, 98, 76 };

    //배열이름 score는 첫 번째 원소의 주소
    printf("score: %p, &score[0]: %p\n", score, &score[0]);

    //배열이름 score는 첫 번째 값
    printf("*score: %d, score[0]: %d\n\n", *score, score[0]);

    printf("첨자   주소     저장값\n");
    //배열이름 score를 사용한 주소와 원소 값 참조
    for (int i = 0; i < SIZE; i++)
        printf("%2d %10p %6d\n", i, (score + i), *(score + i));

    return 0;
}
```

- 배열이름을 사용한 원소값과 주소값의 참조
- 포인터 형변환
  - 주소값: &score[i] == score + i
  - 참조값: score[i] == *(score + i)

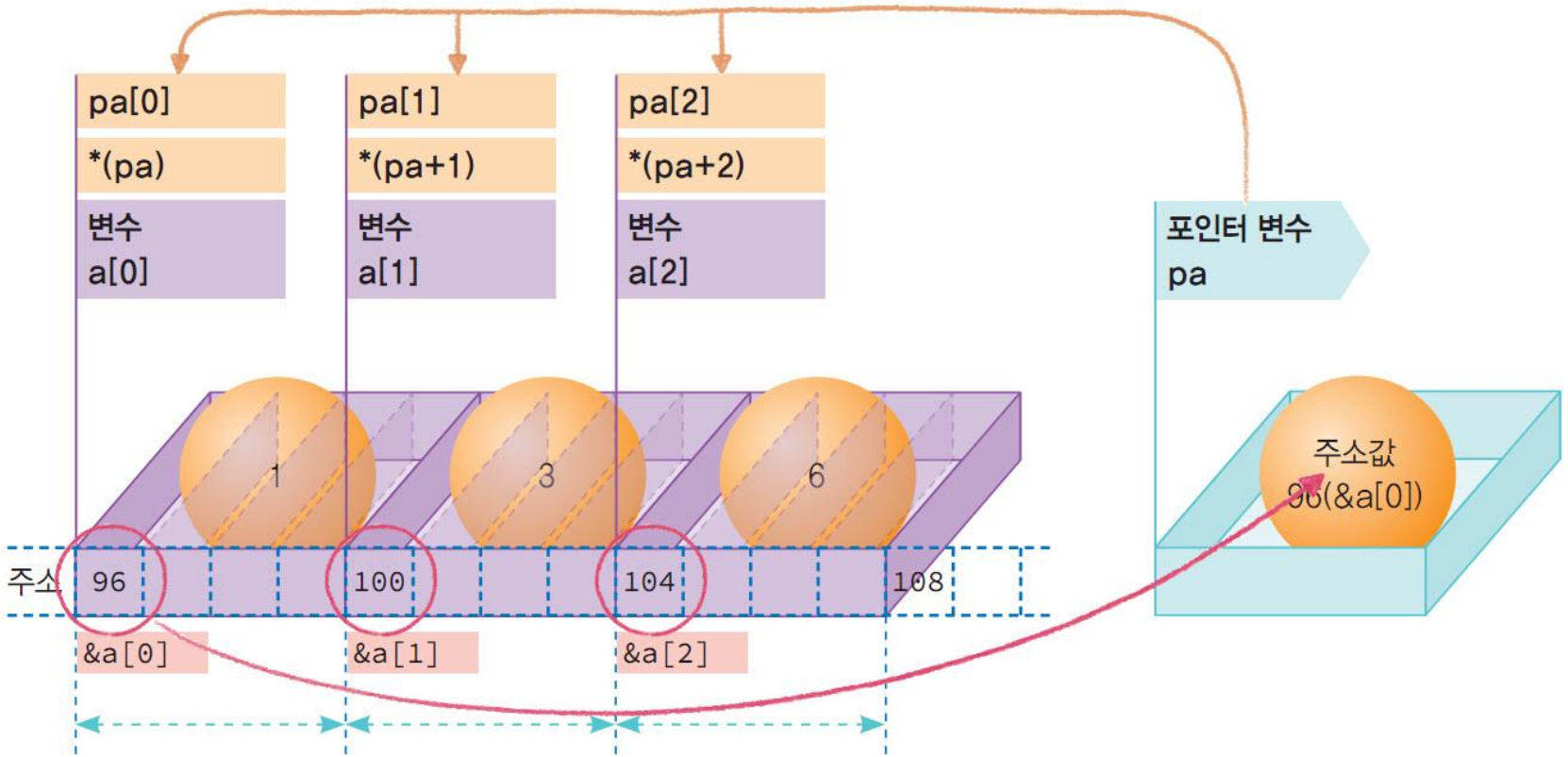
```
score: 00AFFD3C, &score[0]: 00AFFD3C
*score: 89, score[0]: 89

첨자   주소     저장값
 0   00AFFD3C     89
 1   00AFFD40     98
 2   00AFFD44     76
```

포인터 변수를 이용한 배열의 원소 참조

- 배열 첫 원소의 주소를 포인터에 저장한 후
  - 주소를 1씩 증가시키면 각각의 원소를 참조 가능
  - int a[4] = {1, 3, 6};
  - int *pa = &a[0];
  - 포인터 pa에 &a[0]를 저장하면 연산식 *(pa+i)으로 배열원소를 참조 가능
  - 포인터 pa로도 배열과 같이 첨자를 이용 pa[i]로 배열원소를 참조 가능

포인터 변수를 이용한 배열의 원소 참조

```
int a[4] = {1, 3, 6};
int *pa = &a[0];

printf("%d  %d  %d\n", *(pa), *(pa+1), *(pa+2));     //1, 3, 6 출력
printf("%d  %d  %d\n", pa[0], pa[1], pa[2]);         //1, 3, 6 출력
```



포인터의 증감연산과 간접연산자

- 연산자 *, ++, --
  - 참조연산자 *의 우선순위는 ++p의 전위 증감연산자와 같고
  - 괄호나 p++의 후위 증감연산자보다 낮다
  - 연산식 *p++는 *(p++)를 의미
  - *p++: 포인터 p가 가리키는 변수를 참조하고 p의 주소를 1 증가
  - 반면 (*p)++: 포인터 p가 가리키는 변수를 참조하고 그 값을 1 증가
  - 연산식 *++p는 *(++p): 포인터 p를 1 증가시킨 후 가리키는 변수를 참조
  - 연산식 ++*p는 ++(*p): 포인터 p가 가리키는 값을 1 증가시킨 후 참조

연산식		결과값	연산 후 *p의 값	연산 후 p 증가
++*p	++(*p)	*p + 1: p의 간접참조 값에 1 증가	*p가 1 증가	p: 없음
*p++	*(p++)	*p: p의 간접참조 값	변동 없음	p+1: p 다음 주소
--*p	--(*p)	*p - 1: p의 간접참조 값에 1 감소	*p가 1 감소	p: 없음
(*p)--		*p: p의 간접참조 값	*p가 1 감가	p: 없음

명시적 형변환

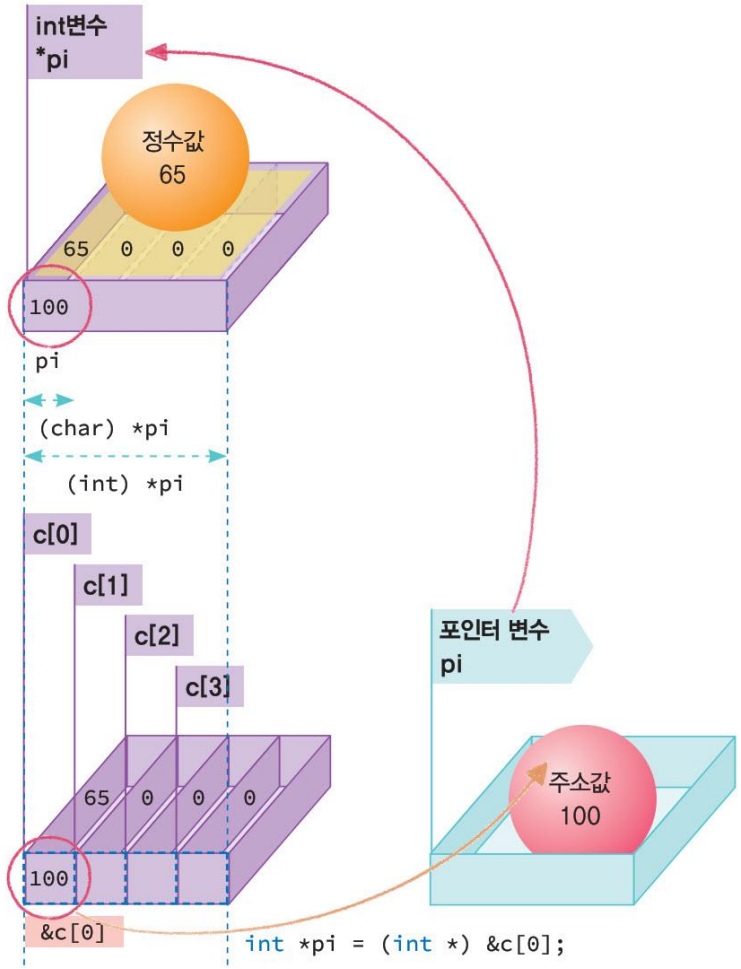
- 포인터 변수는 자동으로 형변환type cast이 불가능
  - 필요하면 명시적으로 형변환을 수행
- 자료형 (char *)를 (int *)로 변환
  - (int *) 형 변수 pi에 저장하는 것은 가능
  - *pi로 수행하는 간접참조
    - pi가 가리키는 주소에서부터 4바이트 크기의 int 형 자료를 참조
    - 문자배열에 저장된 문자 4개를 그대로 4바이트인 정수 65로 참조
  - 변환된 포인터 변수는 지정된 주소값을 시작하여 그 변수 자료형의 크기만큼 저장공간을 참조
    - 동일한 메모리의 내용과 주소로부터 참조하는 값이 포인터의 자료형에 따라 달라진다는 것

```
char c[4] = {'A', '\0', '\0', '\0'}; //문자'A' 코드값: 65

int *pi = &c[0];
```

warning C4133: '초기화중' : 'char *'과(와) 'int *' 사이의 형식이 호환되지 않습니다.

명시적 형변환

```
char c[4] = {'A', '\0', '\0', '\0'}; //문자'A' 코드값: 65
int *pi = (int *) &c[0];
printf("%d %c\n", (int) *pi, (char) *pi); //정수값 65와 문자'A'가 출력
```



Source Code #07: ptypecast.c

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

int main(void)
{
    char c[4] = { 'A', '\0', '\0', '\0' }; //문자'A' 코드값: 65
    //int *pi = &c[0]; //경고 발생
    int *pi = (int *)&c[0];

    printf("%d %c\n", (int)c[0], c[0]);
    printf("%d %c\n", *pi, (char)*pi);

    return 0;
}
```

```
65 A
65 A
```

- 포인터의 간접참조
  - 시작주소에서 포인터 자료형 크기만큼 참조

배열이름과 행이름으로 참조(1)

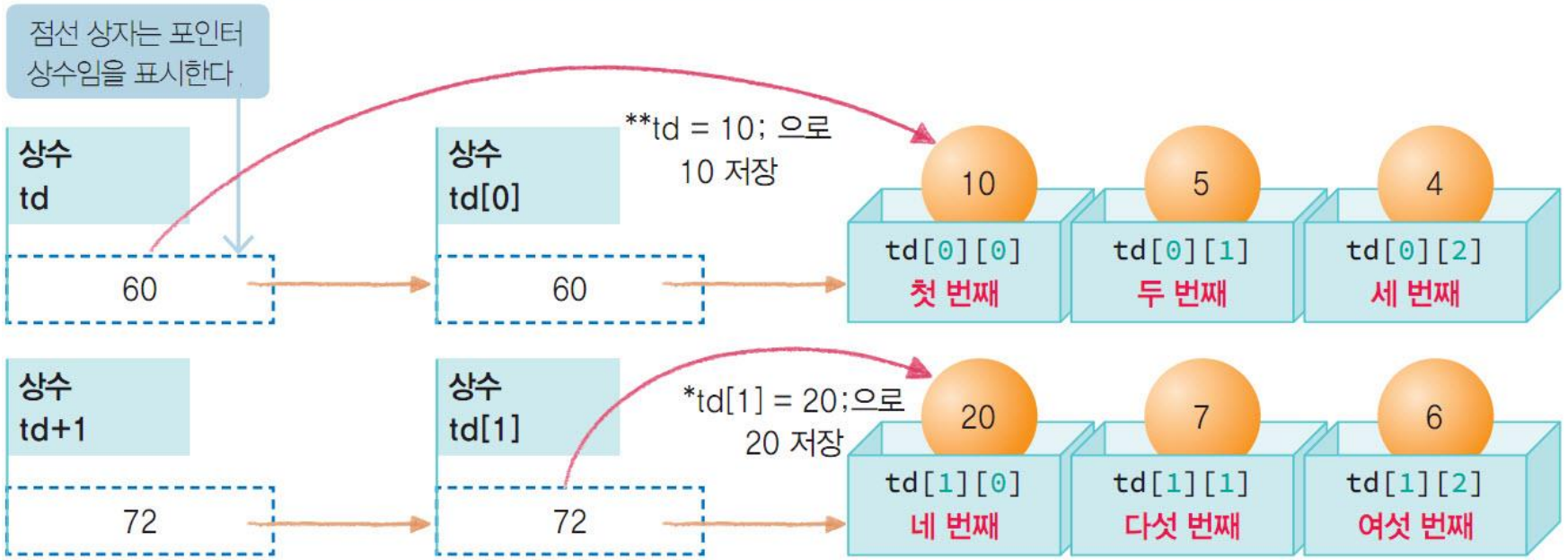
- 이차원 배열에서 배열이름인 td
  - 포인터 상수 td[0]를 가리키는 포인터 상수
  - 포인터 상수 td[0]
  - 배열의 첫 번째 원소 td[0][0]의 주소값, &td[0][0]을 갖는 포인터 상수
  - 배열이름인 td는 포인터의 포인터인 이중 포인터
- int td[][3] = {{8, 5, 4}, {2, 7, 6}};
  - 배열이름 td는 이차원 배열을 대표하는 이중 포인터
  - sizeof(td)는 배열전체의 바이트 크기를 반환
  - 배열이름 td를 이용하여 변수 td[0][0]의 값을 20으로 수정
    - **td = 20;
    - td가 이중 포인터이므로 간접연산자 *이 2개 필요
- td[i]는 (i+1) 번째 행을 대표
  - (i+1) 번째 행의 처음을 가리키는 포인터 상수
- sizeof(td[0])와 sizeof(td[1])
  - 각각 첫 번째 행의 바이트 크기와 두 번째 행의 바이트 크기를 반환
- td[1]은 두 번째 행의 첫 원소의 주소
  - *td[1]로 td[1][0]를 참조 가능

배열이름과 행이름으로 참조(2)

- 연산식 (*td+n)
  - 배열의 (n+1)번째 원소의 주소값
- 연산식 *(*td+n)
  - 배열의 (n+1)번째 원소 자체
- td[i]
  - (i+1) 번째 행의 주소
  - (td[i] + j)는 &td[i][j]
- *(td[i]+j)
  - 배열의 td[i][j] 원소 자체
- 일차원 배열에서 a[i]
  - *(a+i)
- 이차원 배열 td[i][j]
  - j크기의 일차원 배열로 간주
  - *(td[i] + j)
  - a[i] == *(a+i)
- td[i][j]
  - == (td[i]) [j]
  - == * (td[i] + j)
  - == * (*(td + i) + j)

배열이름과 행이름으로 참조(2)

```
int td[][3] = {{8, 5, 4}, {2, 7, 6}};
int i = 0, j = 0, cnt = 0;

printf("%d, %d, %d\n", sizeof(td), sizeof(td[0]), sizeof(td[1]));
printf("%u, %u, %u\n", td, td[0], td[1]);
printf("%u, %u\n", &td[0][0], &td[1][0]);

**td = 10;    //td[0][0] = 10;
*td[1] = 20;    //td[1][0] = 20;
```



배열이름과 행이름으로 참조(2)

```
for ( i = 0; i < ROW; i++ )
{
    for ( j = 0; j < COL; j++, cnt++ )
    {
        printf("%d %d, ", *(*td + cnt), *(td[i] + j));
    }
    printf("\n");
}
```

연산식 (*td + n)은 (n+1)번째 원소의 주소값이므로

```
td[0][0]: *(*td + 0)
td[0][1]: *(*td + 1)
td[0][2]: *(*td + 2)
td[1][0]: *(*td + 3)
td[1][1]: *(*td + 4)
td[1][2]: *(*td + 5)
```

연산식 (td[i] + j)은 td[i][j]의 주소값이므로

```
td[0][0]: *(td[0] + 0)
td[0][1]: *(td[0] + 1)
td[0][2]: *(td[0] + 2)
td[1][0]: *(td[1] + 0)
td[1][1]: *(td[1] + 1)
td[1][2]: *(td[1] + 2)
```

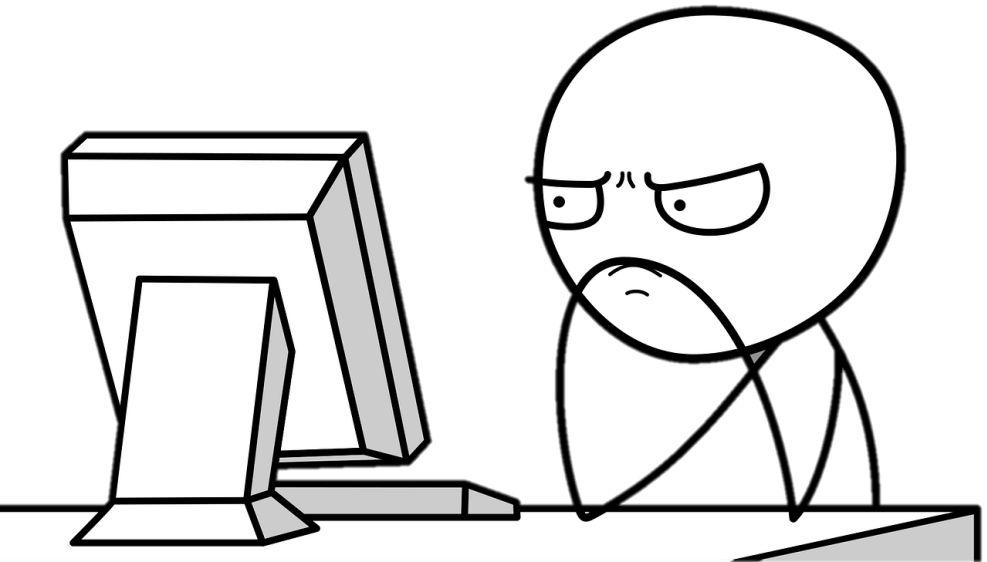
Source Code #08: tdaryptr.c

```c
// fiel: tdaryptr.c
#include <stdio.h>

#define ROW 2
#define COL 3

int main(void)
{
    int td[][COL] = { { 8, 5, 4 }, { 2, 7, 6 } };

    **td = 10;      //td[0][0] = 10;
    *td[1] = 20;    //td[1][0] = 20;

    for (int i = 0, cnt = 0; i < ROW; i++)
    {
        for (int j = 0; j < COL; j++, cnt++)
        {
            printf("%d %d %d, ", *(*td + cnt), *(td[i] + j), *(*(td + i) + j));
        }
        printf("\n");
    }

    printf("%d, %d, %d\n", sizeof(td), sizeof(td[0]), sizeof(td[1]));
    printf("%p, %p, %p\n", td, td[0], td[1]);
    printf("%p, %p\n", &td[0][0], &td[1][0]);

    return 0;
}
```

- 배열이름과 행의 대표이름으로 배열 원소 참조

```
10 10 10, 5 5 5, 4 4 4,
20 20 20, 7 7 7, 6 6 6,
24, 12, 12
0136FBFC, 0136FBFC, 0136FC08
0136FBFC, 0136FC08
```

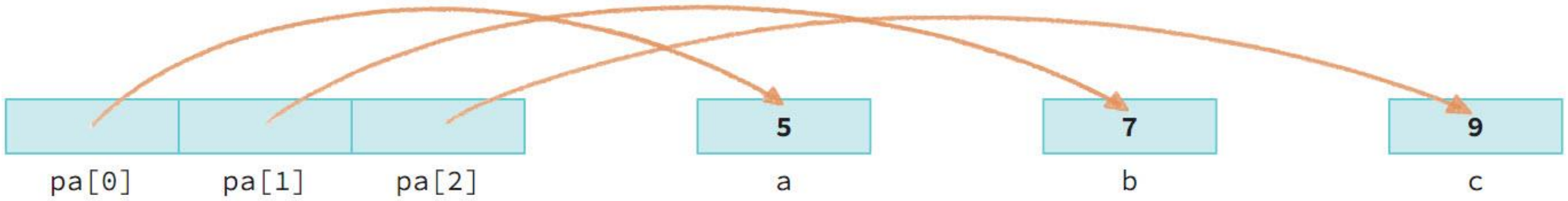
4. 포인터 배열과 배열 포인터



포인터 배열array of pointer

- 주소값을 저장하는 포인터를 배열 원소로 하는 배열
- 다음 pa는 배열크기가 3인 포인터 배열
  - pa[0]는 변수 a의 주소를 저장
  - pa[1]는 변수 b의 주소
  - pa[2]는 변수 c의 주소

```
int a = 5, b = 7, c = 9;

int *pa[3];

pa[0] = &a;    pa[1] = &b;   pa[2] = &c;
```



포인터 배열array of pointer

- 문장 double *dary[5] = {NULL};
  - 나머지 모든 배열원소에 NULL 주소가 저장

포인터 배열 변수선언

자료형 *변수이름[배열크기] ;

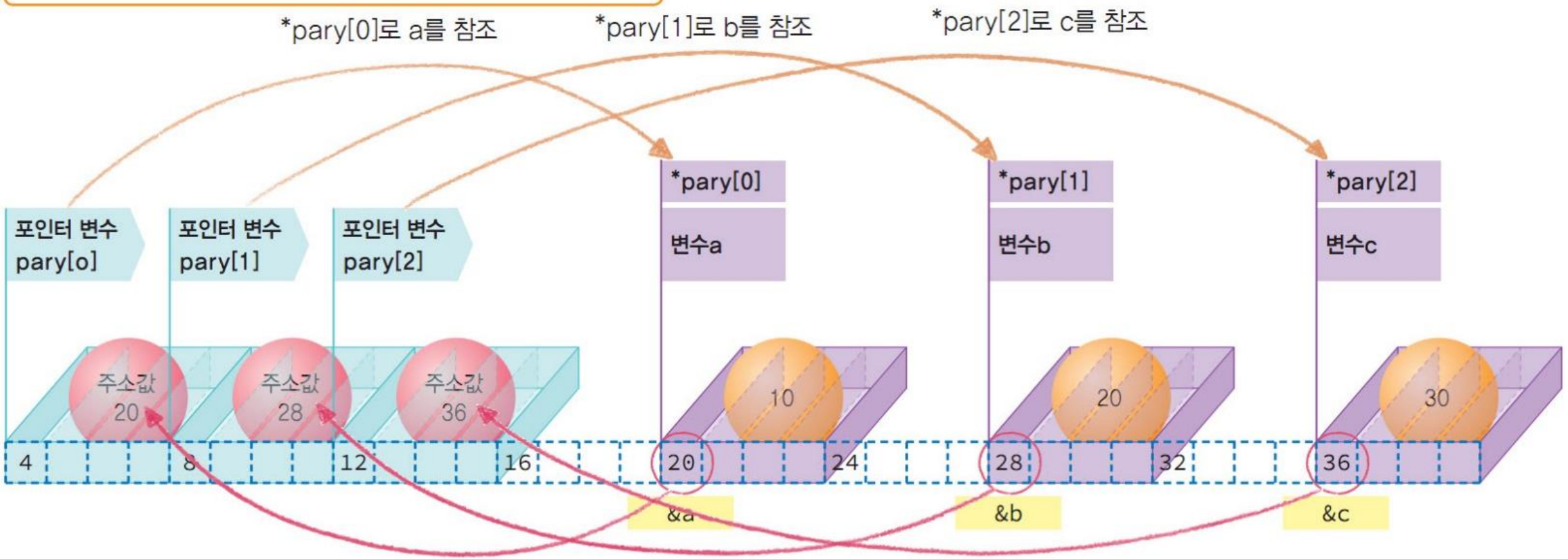
```
int *pary[5];
char *ptr[4];
float a, b, c;
double *dary[5] = {NULL};
float *ptr[3] = {&a, &b, &c};
```

포인터 배열array of pointer

- 포인터 배열 pary는 int형 포인터 3개를 원소로 갖는 배열
  - 배열 pary는 선언하면서 초기값으로 모두 NULL로 저장
  - 이후 포인터 배열 pAry의 각각의 원소에 변수 a, b, c의 주소를 저장
  - 이제 역참조에 의해 *pary[0]은 변수 a를 참조 가능
  - 마찬가지로 *pary[1] 은 변수 b를 참조
  - *pary[2]는 변수 c 를 참조

포인터 배열array of pointer

```
int *pary[3] = { NULL };
int i, a=10, b=20, c=30;

pary[0] = &a;  pary[1] = &b; pary[2] = &c;
```



Source Code #09: pointerarray.c

```c
// file: pointerarray.c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#define SIZE 3

int main(void)
{
    //포인터 배열 변수선언
    int *pary[SIZE] = { NULL };
    int a = 10, b = 20, c = 30;

    pary[0] = &a;
    pary[1] = &b;
    pary[2] = &c;
    for (int i = 0; i < SIZE; i++)
        printf("*pary[%d] = %d\n", i, *pary[i]);

    for (int i = 0; i < SIZE; i++)
    {
        scanf("%d", pary[i]);
        printf("%d, %d, %d\n", a, b, c);
    }

    return 0;
}
```

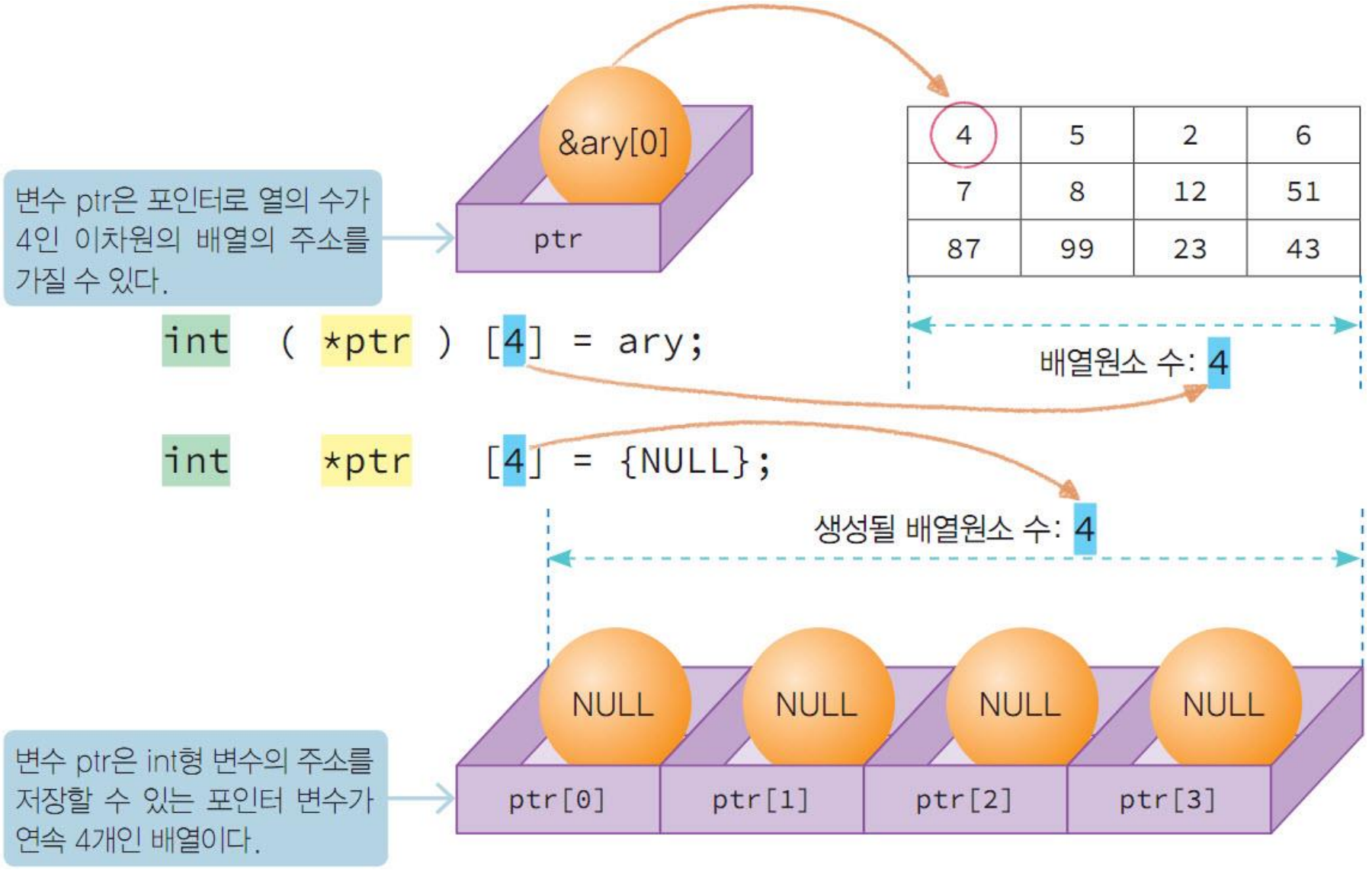
- 포인터 배열 pary를 이용해 표준입력을 받아 다시 원래 변수 a, b, c로 출력
- 반복문 내부 scanf()에서 표준입력값이 저장되는 실인자로 pary[i]를 사용하는 것에 주의
- 일반변수라면 주소연산자 &가 앞에 필요하지만 pary[i]가 저장할 주소이므로 그대로 사용

```
*pary[0] = 10
*pary[1] = 20
*pary[2] = 30
21
21, 20, 30
15
21, 15, 30
8
21, 15, 8
```

이차원 배열 포인터pointer to array 선언

- 자료형 int인 일차원 배열 int a[]의 주소
  - (int *)인 포인터 변수에 저장 가능
- 이차원 배열 ary[][4]의 주소를 저장
  - 배열 포인터 변수 ptr
    - 문장 int (*ptr)[4];로 선언
    - 4는 가리키는 이차원 배열에서의 열 크기
    - 포인터 변수는 열 크기에 따라 변수 선언이 달라짐
    - 괄호 (*ptr)은 반드시 필요
  - 괄호가 없는 int *ptr[4];
    - 바로 전에 배운 int형 포인터 변수 4개를 선언하는 포인터 배열 선언문장

이차원 배열 포인터pointer to array 선언



이차원 배열 포인터pointer to array 선언

일차원과 이차원 배열 포인터의 변수 선언

원소자료형 *변수이름;
변수이름 = 배열이름;
또는
원소자료형 *변수이름 = 배열이름;

```
int a[] = {8, 2, 8, 1, 3};
int *p = a;
```

원소자료형 (*변수이름)[배열열크기];
변수이름 = 배열이름;
또는
원소자료형 (*변수이름)[배열열크기] = 배열이름;

```
int ary[][4] = {5, 7, 6, 2, 7, 8, 1, 3};
int (*ptr)[4] = ary;  //열이 4인 배열을 가리키는 포인터
//int *ptr[4] = ary;  //포인터 배열
```

일차원 배열 포인터 활용

- 배열이름 a는 포인터 상수
- 변수 p는 포인터 변수
- 그러나 p와 a 모두 배열 a 첫 원소의 주소값을 가짐
- p는 변수
  - p++ 또는 ++p 연산이 가능
  - sizeof(p)은 단순히 포인터의 크기인 4
- a는 상수
  - a++ 또는 ++a 연산을 수행 불가능
  - sizeof(a)는 배열의 총 바이트 크기인 20 = (5*4)
- 배열이름이 대입된 배열 포인터 변수 p
  - a[i]와 같이 p[i]로 배열 a의 모든 원소를 참조

일차원 배열 포인터 활용

```c
int a[] = {8, 2, 8, 1, 3};
int *p = a;

printf("%2d, %2d\n", *(p+1), *(p+4));
printf("%2d, %2d\n", p[0], p[4]);
printf("%2d, %2d\n", sizeof(a), sizeof(p));

printf("%2d\n", *++p); // 배열의 두 번째 원소 참조
//printf("%2d\n", *++a);  //오류 발생
```

이차원 배열 포인터 활용

- 이차원 배열 소스에서 변수이름 ary는 배열을 대표하는 배열이름
- 변수 ptr은 열의 수가 4인 이차원 배열의 주소값을 저장할 수 있는 배열 포인터
- 변수 ptr은 배열 ary 첫 원소의 주소값
  - 배열 첫 원소를 참조하려면 **ptr을 이용
- 연산식 **ptr++
  - **(ptr++)와 같으며 현재 포인터가 가리키는 원소를 참조하고 ptr을 하나 증가시켜 다음 원소를 가리키게 하는 연산식
- *(ary[i] + j)와 *(ptr[i] + j)
  - *(*(ary + i) + j)와 *(*(ptr + i) + j)
  - 모두 (i+1) 행, (j+1)열의 원소를 참조
- sizeof(ary)는 배열의 총 크기인 32 = (4*2*4)
  - sizeof(ptr)은 단순히 포인터의 크기인 4

이차원 배열 포인터 활용

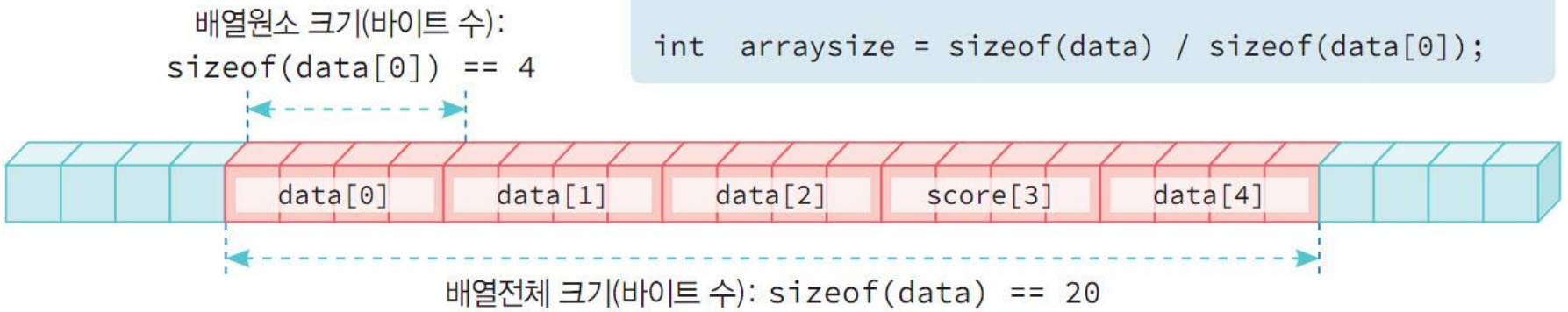
```
int ary[][4] = {5, 7, 6, 2, 7, 8, 1, 3};
int (*ptr)[4] = ary;    //열이 4인 배열을 가리키는 포인터
//int *ptr[4] = ary;    //포인터 배열

printf("%2d, %2d\n", **ary, **ptr++);
printf("%2d, %2d\n", **(ary+1), **(ptr++));
ptr = ary;
printf("%2d, %2d\n", *(ary[1] + 1), *(ptr[1] + 1));
printf("%2d, %2d\n", *(*(ary+1) + 3), *(*(ptr+1) + 3));
printf("2%d, %2d\n", sizeof(ary), sizeof(ptr));
```

일차원 배열크기 연산

- 저장공간의 크기를 바이트 수로 반환하는 연산자 sizeof를 이용
- (sizeof(배열이름) / sizeof(배열원소))의 결과는 배열크기
  - sizeof(배열이름)은 배열의 전체 공간의 바이트 수
  - sizeof(배열원소)는 배열원소 하나의 바이트 수

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

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

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

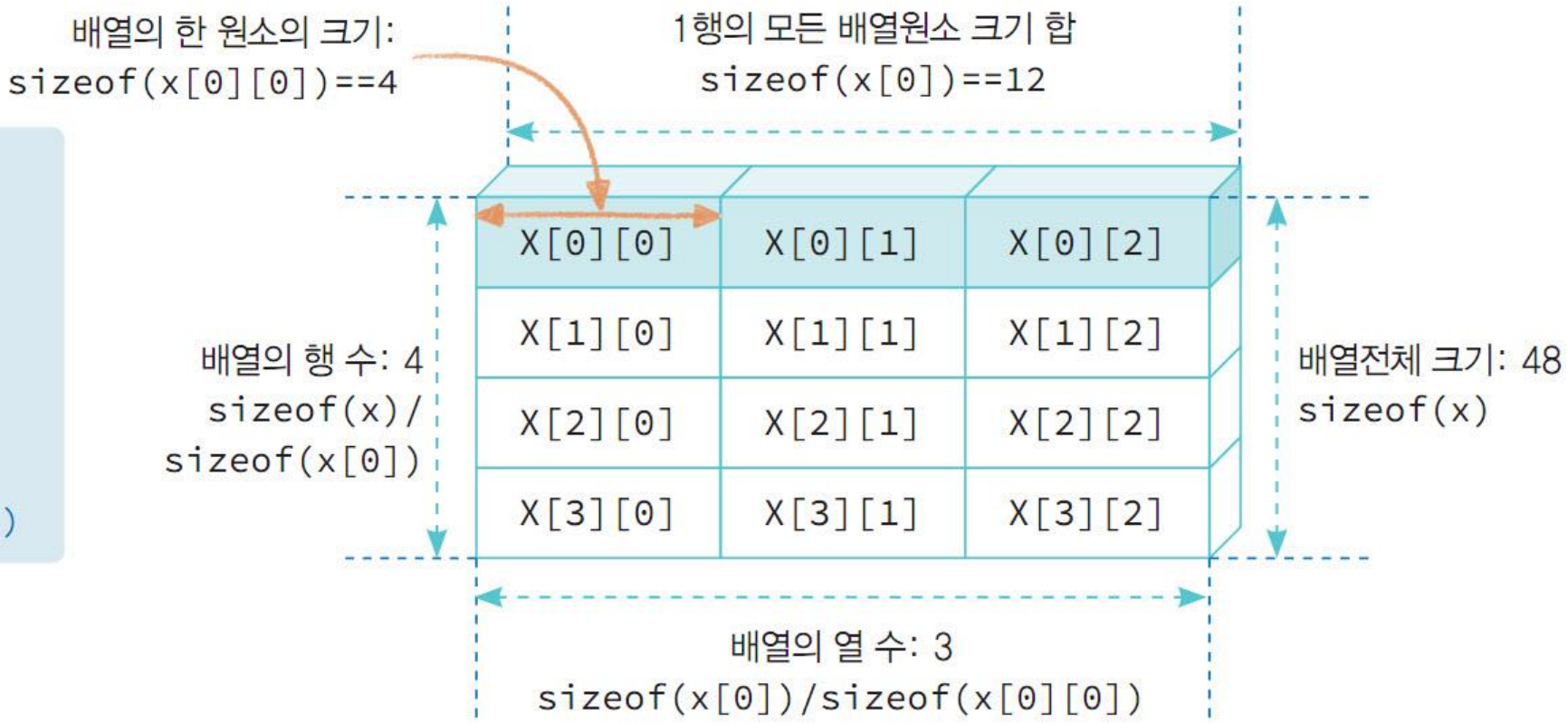


이차원 배열크기 계산방법

- 이차원 배열 x[][]
- 이차원 배열의 행의 수
  - (sizeof(x) / sizeof(x[0]))로 계산
  - sizeof(x): 배열 전체의 바이트 수
  - sizeof(x[0]): 1행의 바이트 수
- 이차원 배열의 열의 수
  - (sizeof(x[0]) / sizeof(x[0][0]))로 계산
  - sizeof(x[0][0]): 첫 번째 원소의 바이트 수

이차원 배열크기 계산방법

```
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])
```



Lab #03: 각 행의 첫 주소와 2행의 모든 원소의 주소와 값을 출력

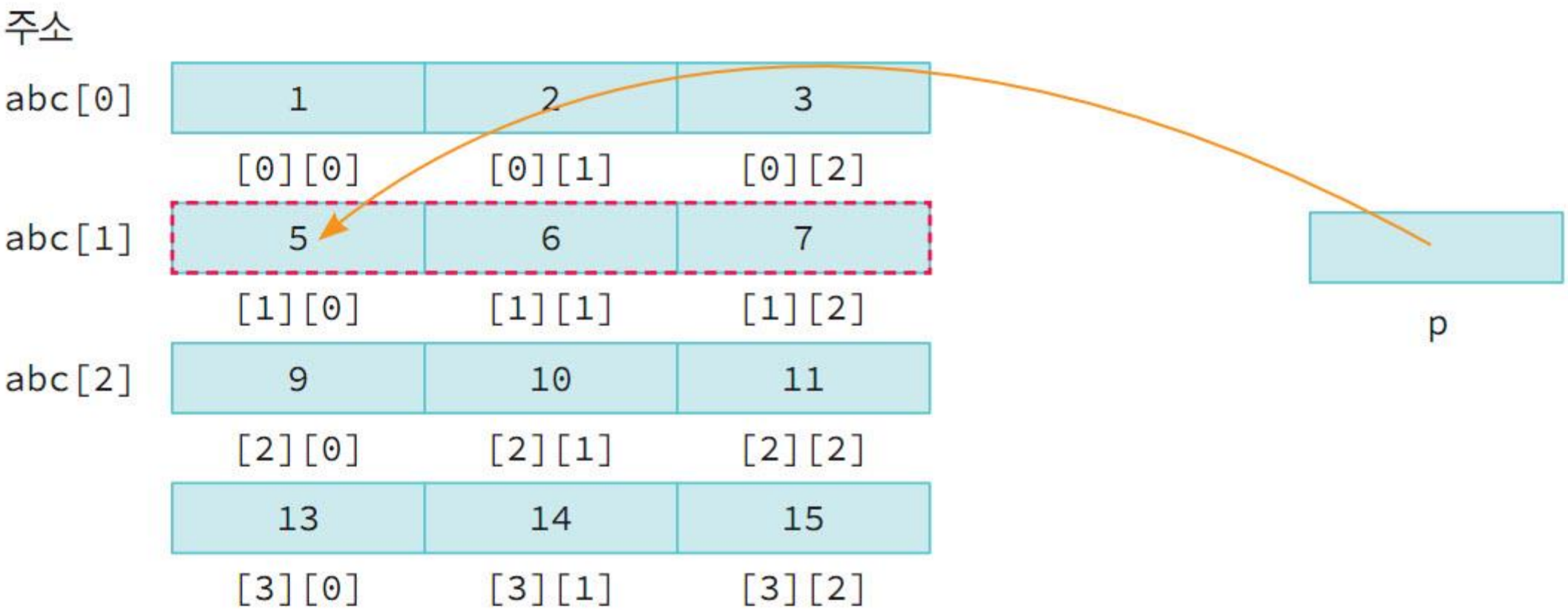
- 자료형 int로 선언된 이차원 배열 abc에서 각 행의 첫 주소와 2행의 모든 원소의 주소와 값을 출력
- 필요한 행수와 열수
  - 각각 변수 rowsize와 colsize에 연산식을 사용해 저장
- 각 행의 주소를 표현하는 abc[i]를 고려하여 코딩
- 각 행만을 생각한다면 각 행을 일차원 배열로 간주

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

int main()
{
    int abc[4][3] = {
        { 1, 2, 3 },
        { 5, 6, 7 },
        { 9, 10, 11 },
        { 13, 14, 15 }
    };
    int rowsize = ;
    int colsize = ;

    printf("각 행의 첫 주소 출력: \n");
    for (int i = 0; i < rowsize; i++)
        printf("%p ", );
    printf("\n\n");

    printf("2행 원소의 주소와 값 출력: \n");
    int *p = abc[1];
    for (int i = 0; i < colsize; i++)
    {
        printf("%p ", p);
        printf("%d\n", );
    }

    return 0;
}
```

```
각 행의 첫 주소 출력:
012FFA98 012FFAA4 012FFAB0 012FFABC

2행 원소의 주소와 값 출력:
012FFAA4 5
012FFAA8 6
012FFAAC 7
```

Lab #03: 각 행의 첫 주소와 2행의 모든 원소의 주소와 값을 출력

```
int *p = abc[1];
```




I
PROGRAMMING