제09장

배 열

Perfect C

C언어로 배우는 프로그래밍 기초

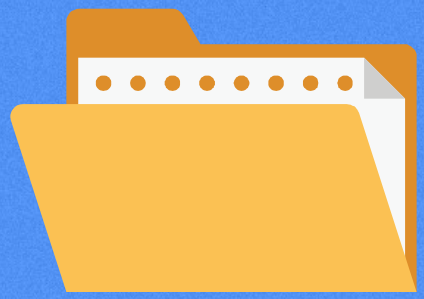
단원 목표

학습목표

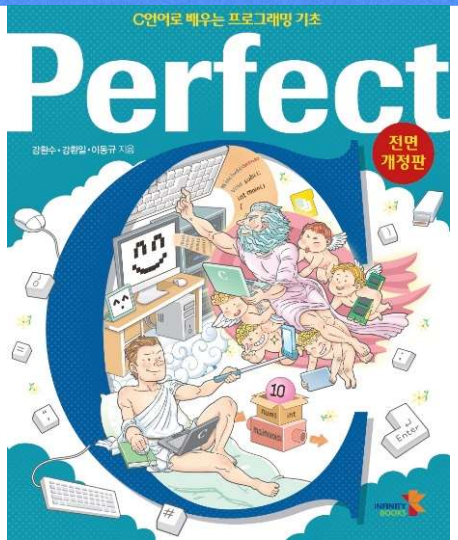
- **배열의 개요와 배열선언 구문에 대하여 이해하고 설명할 수 있다.**
  - 여러 자료의 처리에 필요한 자료구조
  - 자료형, 배열이름, 배열크기를 이용한 배열선언
  - 생성된 배열에서 원하는 원소를 참조
  - 배열선언 시 동시에 초기값 지정 방법
  - 배열선언 초기값 설정에서 배열크기 관계
  - 배열에서의 기본값과 쓰레기값
- **다차원 배열에 대하여 다음을 이해하고 설명할 수 있다.**
  - 이차원 배열의 개념과 선언 방법
  - 이차원 배열의 배열선언 초기값 설정
  - 삼차원 배열의 개념과 배열선언과 초기값 설정
- **배열과 포인터 관계에 대하여 이해하고 설명할 수 있다.**
  - 배열이름은 포인터 상수이며 포인터 변수로도 참조
  - int 자료형을 char 저장공간으로 활용
  - double 자료형을 int 저장공간으로 활용
  - 포인터 배열과 배열 포인터를 구분
  - 일차원과 이차원 배열에서 여러 크기를 계산하는 연산식

학습목차

9.1 배열 선언과 초기화
9.2 이차원과 삼차원 배열
9.3 배열과 포인터 관계
9.4 포인터 배열과 배열 포인터



01. 배열 선언과 초기화



배열의 필요성과 정의

- **배열(array)의 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 원소(elements)
      - 배열원소는 첨자(index) 번호라는 숫자를 이용해 쉽게 접근
    - 배열이름, 원소 자료유형, 배열크기

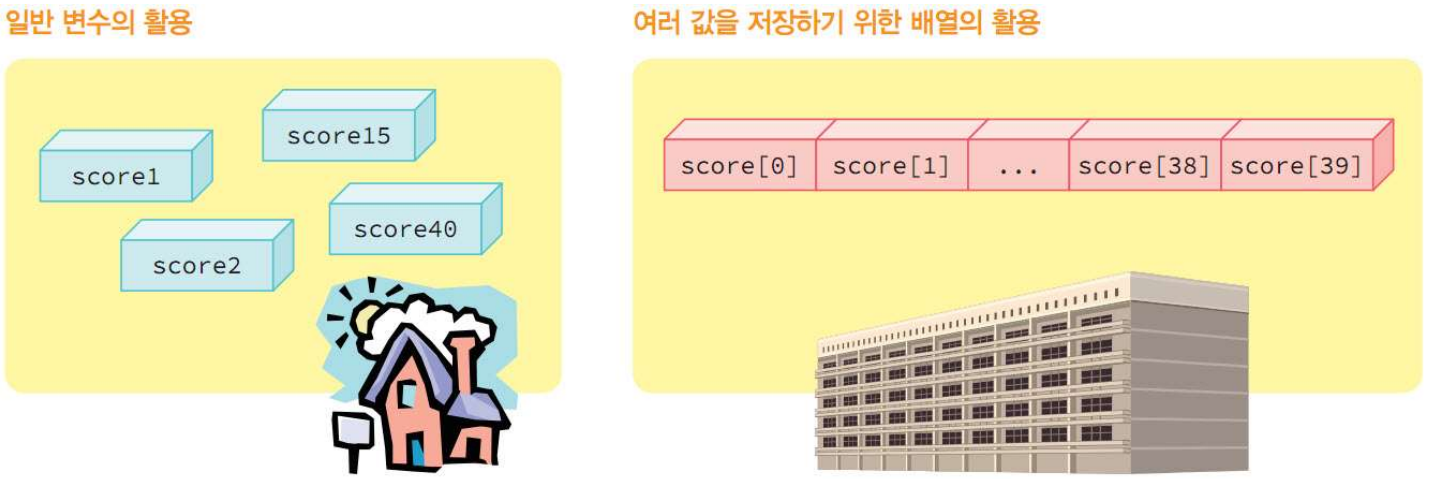


그림 9-1 배열의 필요성과 개념

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

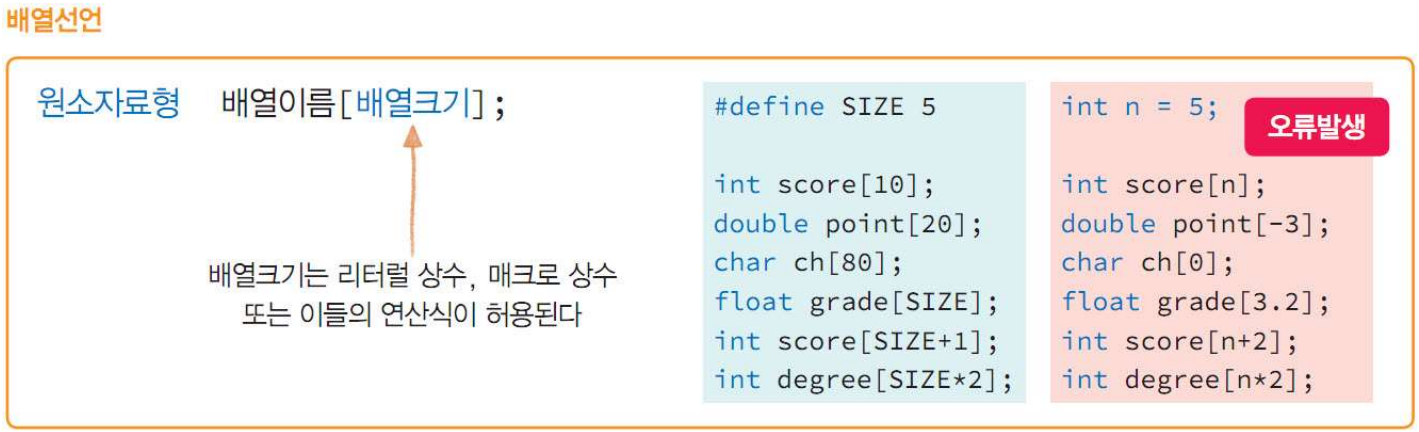
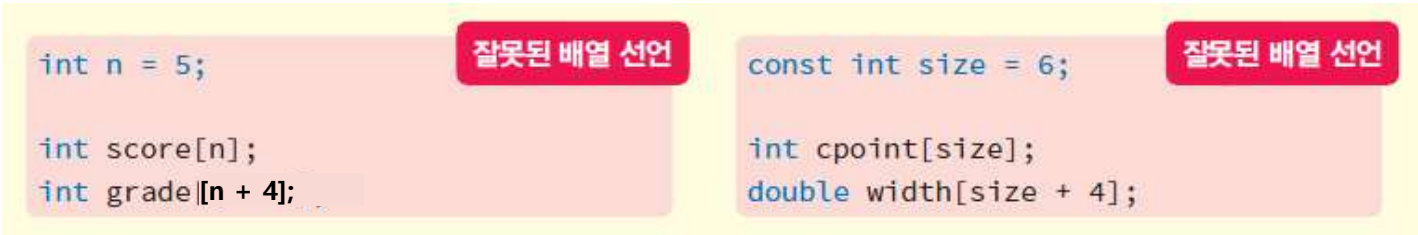


그림 9-2 배열선언 구문



배열원소 접근

- **배열선언 후 배열원소에 접근**
  - 배열이름 뒤에 대괄호 사이 첨자(index)를 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

```
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
```

그림 9-3 배열선언과 원소참조

배열원소 초기값

- **초기값**
  - 배열을 함수내부에서 선언 후 원소에 초기값을 저장하지 않으면
  - 쓰레기 값이 저장되니 항상 초기값을 저장

```
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
```

그림 9-3 배열선언과 원소참조

배열원소 일괄 출력

예제 declarearray.c

- 배열 선언과 모든 원소 출력

제어변수

- 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

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

첨자가 0에서 SIZE-1까지 순회해야 하므로 이 조건을 이용함

그림 9-4 배열원소 출력을 위한 반복 구문

실습예제 9-1 declarearray.c

배열선언 후 배열원소에 값을 저장한 후 순차적으로 출력

```
01  // file: declarearray.c
02  #include <stdio.h>
03
04  #define SIZE 5
05
06  int main(void)
07  {
08     //배열선언
09     int score[SIZE];//int score[5];
10
11     //배열 원소에 값 저장
12     score[0] = 78;
13     score[1] = 97;
14     score[2] = 85;
15     //배열 4번째 원소에 값 저장하지 않아 쓰레기값 저장
16     score[4] = 91;
17     //score[5] = 50; //문법오류는 발생하지 않으나 실행오류 발생
18
19     //배열원소 출력
20     for (int i = 0; i < SIZE; i++)
21        printf("%d  ", score[i]);
22     printf("\n");
23
24     return 0;
25  }
```

설명

```
09     SIZE는 리터럴 상수 또는 매크로 상수로 양의 정수이어야 함
12     배열에서 첫 번째 원소의 첨자는 0이므로 score[0]에 78 저장
12~16  score[0], score[1], score[2], score[4]에 값 저장, score[3]에는 값을
       저장하지 않아 출력 시 쓰레기값 출력
17     첨자가 0에서 4를 벗어나면 실행 오류가 발생
20     반복문에서 제어문자 i를 첨자로 사용
21     반복의 몸체가 printf() 문장 하나로 배열원소를 참조하기 위해 제어문자 i를 첨자로
       하여 score[i]로 기술
22     이 출력문은 for 반복 몸체가 아니므로 들여쓰기에 신경
```

실행결과

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

초기값을 저장하지 않아 쓰레기값이 출력됨

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

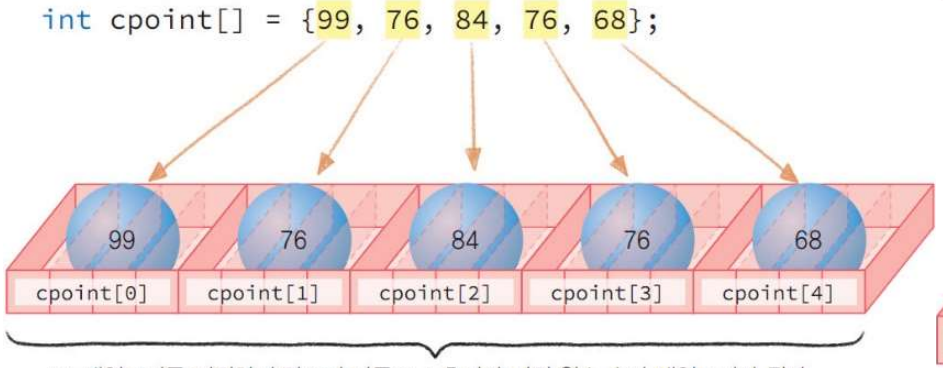
배열선언 초기화

```
원소자료형  배열이름[배열크기] = {원소값1, 원소값2, 원소값3, 원소값4, 원소값5, ... } ;
```

배열크기는 생략 가능하며, 생략 시 원소값의 수가 배열크기가 된다.

```
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};
```

그림 9-5 배열 초기화 구문



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

그림 9-7 배열크기가 없는 경우 배열크기는 자동으로 지정

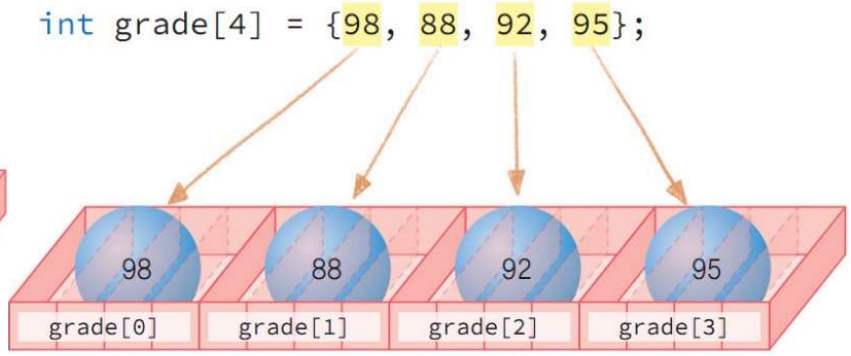


그림 9-6 배열크기가 지정된 경우의 배열원소 초기값 지정

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

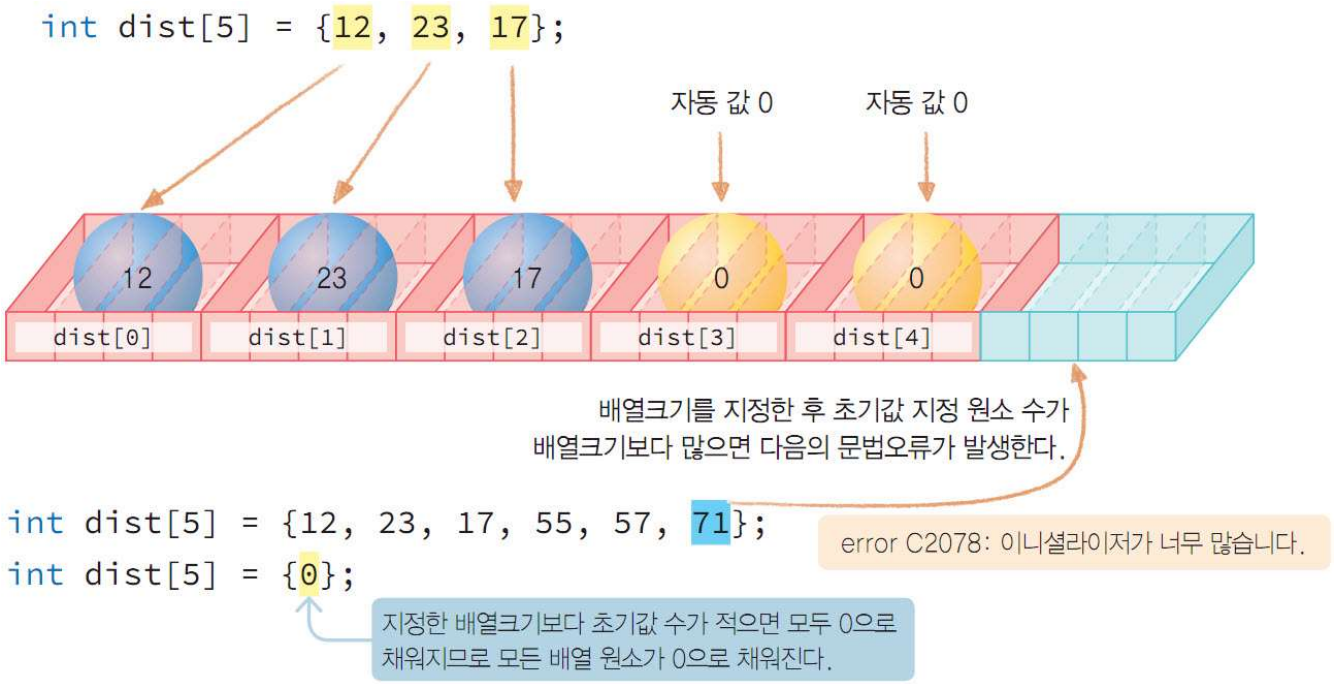


그림 9-8 배열크기가 지정된 경우의 초기값 수

배열선언 초기화

예제 initarray.c

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

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};
```

실습예제 9-2 initarray.c

배열선언 초기화를 이용한 합과 평균 출력

```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;
}
```

설명

```
SIZE는 배열크기인 매크로 상수로 양의 정수인 6으로 정의, 12행의 for 문의 조건에 사용
배열 score의 초기화에서 배열크기를 지정하지 않고 6개의 점수를 직접 기술
모든 점수의 합이 저장될 변수 sum 선언하고 0 저장
반복문에서 제어문자 i를 첨자로 사용하여 0에서 5까지 반복
변수 sum에 점수의 합을 계속 추가
배열원소의 첨자와 저장값을 각각 출력, printf() 문장에서 배열원소를 참조하기 위해 제어문자
i를 첨자로 하여 score[i]로 기술
성적의 합과 평균을 출력
```

실행결과

```
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 520.050이고 평균은 86.670이다.
```

LAB 정수 int 형 배열에 표준입력으로 받은 정수를 저장하여 출력

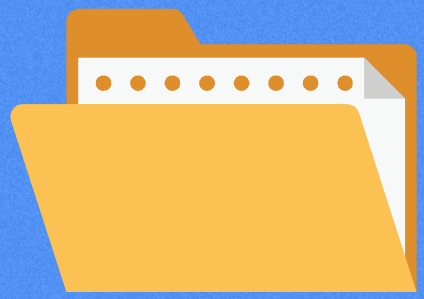
- **자료형 int로 선언된 배열 input에서 표준입력으로 받은 정수를 순서대로 저장하여 출력**
  - 배열 input을 선언하면서 초기화로 모두 0을 저장
  - 표준입력으로 받은 정수는 0이 입력될 때까지 저장
  - 입력된 배열은 while 문을 사용하여 마지막 0 값 이전까지 다음 결과와 같이 출력
    - 배열에 저장할 정수를 여러 개 입력하시오. 0을 입력하면 입력을 종료합니다.
    - 30 26 65 39 87 76 0
    - 30 26 65 39 87 76

Lab 9-1 inputarray.c

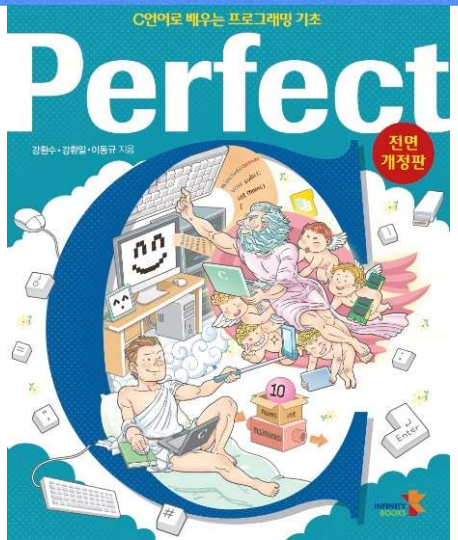
```
01  // file: inputarray.c
02  #define _CRT_SECURE_NO_WARNINGS
03  #include <stdio.h>
04
05  int main(void)
06  {
07     //초기화로 모든 원소에 0을 저장
08     int input[20] = ________________;
09
10     printf("배열에 저장할 정수를 여러 개 입력하시오.");
11     printf(" 0을 입력하면 입력을 종료합니다.\n");
12     int i = 0;
13     do {
14        scanf("%d", ____________________);
15     } while (input[i++] != 0);
16
17     i = 0;
18     while (input[i] != 0) {
19        printf("%d ", ___________________);
20     }
21     puts("");
22
23     return 0;
24  }
```

정답

```
08  int input[20] = { 0 };
14     scanf("%d", &input[i]);
19     printf("%d ", input[i++]);
```

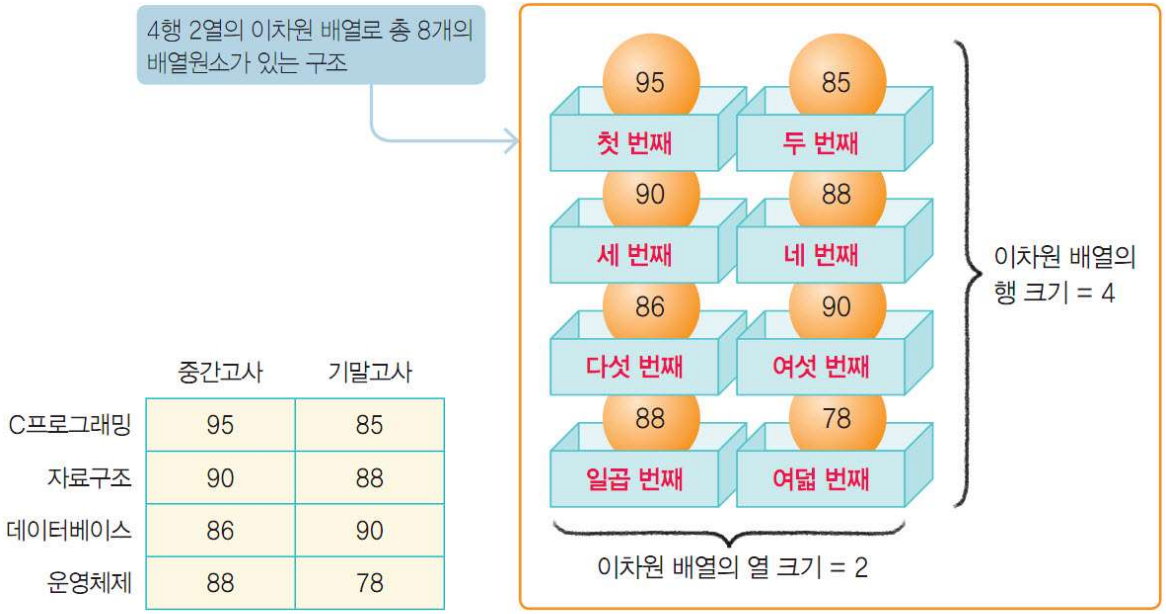


02. 이차원과 삼차원 배열



이차원 배열 선언과 사용

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



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

그림 9-10 이차원 배열의 구조

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];
```

그림 9-11 이차원 배열선언 구문

이차원 배열 구조

원소를 참조하기 위해서는 2개의 첨자가 필요

프로그램

- 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;
```

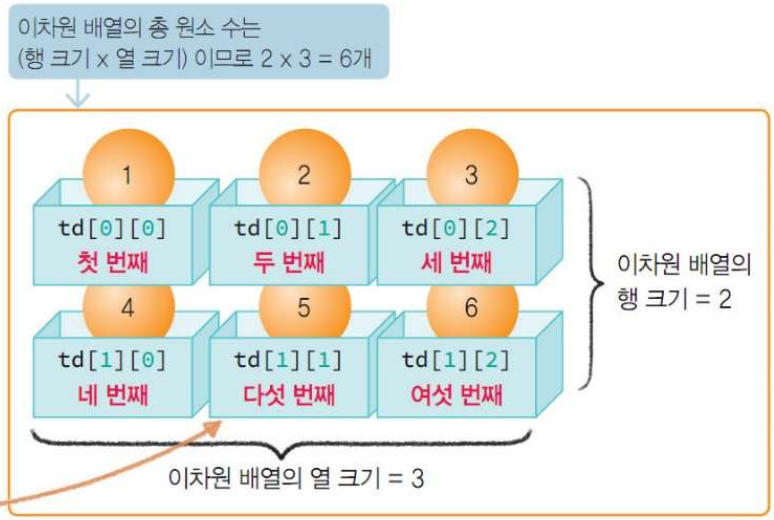


그림 9-12 이차원 배열선언과 구조

행 우선 배열(row major)

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

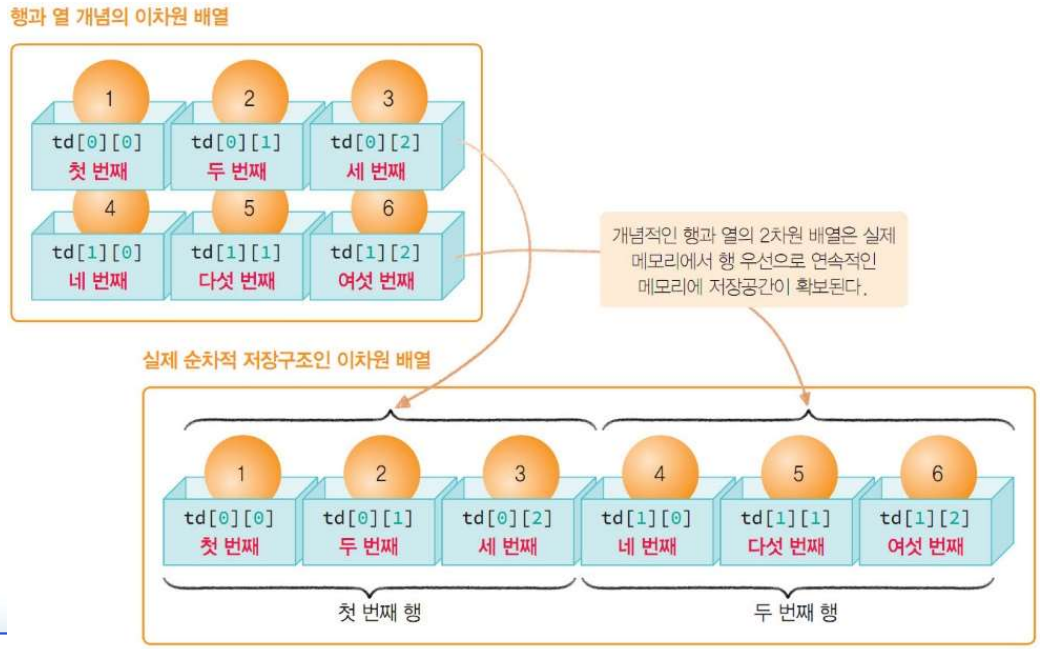


그림 9-13 이차원 배열의 메모리 내부 구조

이차원 배열 원소 참조

예제 twodarray.c

- 이차원 배열 저장과 출력

이차원 배열 출력

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

외부반복 제어변수 i는 행을 순차적으로 참조

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

내부반복 제어변수 j는 한 행에서 열을 순차적으로 참조

그림 9-14 이차원 배열에서 배열원소 순차적으로 출력

실습예제 9-3 twodarray.c

이차원 배열선언과 초기값 직접 저장 후 출력

```
01  // file: twodarray.c
02  #include <stdio.h>
03
04  #define ROWSIZE 2
05  #define COLSIZE 3
06
07  int main(void)
08  {
09     // 2차원 배열선언
10     int td[ROWSIZE][COLSIZE];
11
12     // 2차원 배열원소에 값 저장
13     td[0][0] = 1; td[0][1] = 2; td[0][2] = 3;
14     td[1][0] = 4; td[1][1] = 5; td[1][2] = 6;
15
16     printf("반목문 for를 이용하여 출력\n");
17     for (int i = 0; i < ROWSIZE; i++)
18     {
19        for (int j = 0; j < COLSIZE; j++)
20           printf("td[%d][%d] == %d ", i, j, td[i][j]);
21        printf("\n");
22     }
23
24     return 0;
25  }
```

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

위 반복문으로 대체 가능함.

설명

04 ROWSIZE는 이차원배열 행 크기인 매크로 상수로 양의 정수인 2로 정의, 17행의 for 문의 조건에 사용
05 COLSIZE는 이차원배열 열 크기인 매크로 상수로 양의 정수인 3으로 정의, 19행의 for 문의 조건에 사용
10 배열 td를 2행 3열의 이차원 배열로 선언
13 배열 td의 1행의 각 원소를 1, 2, 3으로 저장
14 배열 td의 2행의 각 원소를 4, 5, 6으로 저장
16 출력 내용 출력
17 반복문에서 제어문자 i를 첨자로 사용하여 0에서 1까지 반복하며, 이 반복은 내부반복 for와 printf()의 두 문장으로 구성되므로 18행과 22행의 중괄호는 반드시 필요
19 내부 반복문에서 제어문자 j를 첨자로 사용하여 0에서 2까지 반복하며, 이 반복은 20행 printf()의 한 문장으로 구성되므로 중괄호는 필요 없으며, 출력 내용은 i, j와 배열 저장값인 td[i][j]
21 한 행을 모두 출력한 이후에 다음 줄로 이동하기 위한 출력으로 들여쓰기에 주의

실행결과

```
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 배열선언 초기화

- **이차원 배열을 선언하면서 초기값을 지정하는 방법 2가지**
  - 중괄호를 중첩되게 이용하는 방법
  - 하나의 중괄호를 사용하는 방법
- **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
- **총 배열원소 수보다 적게 초기값이 주어지면**
  - 나머지는 모두 기본값인 0, 0.0 또는 '₩0'이 저장

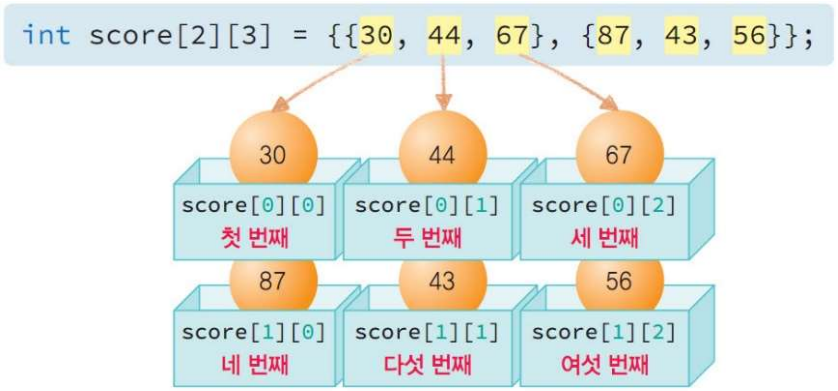


그림 9-16 이차원 배열선언 초기화

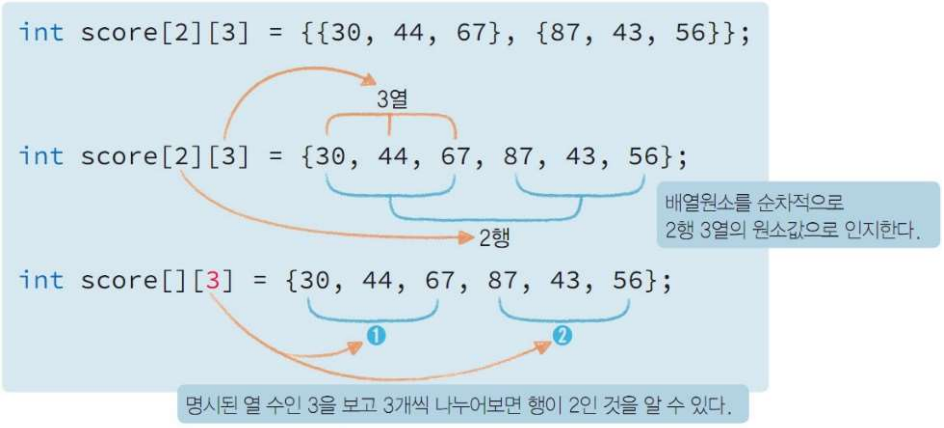


그림 9-17 이차원 배열선언 초기화

```
int a[2][4] = {10, 30, 40, 50, 1, 3, 0, 0};
int a[2][4] = {10, 30, 40, 50, 1, 3};
int a[][4] = {10, 30, 40, 50, 1, 3};
int a[2][4] = { {10, 30, 40, 50}, {1, 3} };
int a[][4] = { {10, 30, 40, 50}, {1, 3} };
```

그림 9-18 이차원 배열 2행 4열에 순차적으로 10, 30, 40, 50, 1, 3, 0, 0이 저장된 초기화 문장

이차원 배열 초기화

예제 inittwodarray.c

- 이차원 배열 초기화와 활용

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};
```

실습예제 9-4 inittwodarray .c

이차원 배열 초기화와 원소 출력

```
01  // file: inittwodarray.c
02  #include <stdio.h>
03
04  #define ROWSIZE 2
05  #define COLSIZE 3
06
07  int main(void)
08  {
09     // 2차원 배열 초기화
10     int td[][3] = { { 1 }, { 1, 2, 3 } };
11
12     printf("반목문 for를 이용하여 출력\n");
13     for (int i = 0; i < ROWSIZE; i++)
14     {
15        for (int j = 0; j < COLSIZE; j++)
16           printf("%d ", td[i][j]);
17        printf("\n");
18     }
19
20     return 0;
21  }
```

설명

04 ROWSIZE는 이차원배열 행 크기인 매크로 상수로 양의 정수인 2로 정의, 13행의 for 문의 조건에 사용

05 COLSIZE는 이차원배열 열 크기인 매크로 상수로 양의 정수인 3으로 정의, 15행의 for 문의 조건에 사용

10 배열 td를 3열의 이차원 배열로 선언하면서 초기값을 저장, 초기값 { { 1 }, { 1, 2, 3 } } 을 통하여 2행임을 알 수 있으며, 1행의 각 원소를 1, 0, 0으로 저장, 2행의 각 원소를 1, 2, 3으로 저장

12 출력 내용 출력

13 반복문에서 제어문자 i를 첨자로 사용하여 0에서 1까지 반복하며, 이 반복은 내부반복 for와 printf()의 두 문장으로 구성되므로 14행과 18행의 중괄호는 반드시 필요

19 내부 반복문에서 제어문자 j를 첨자로 사용하여 0에서 2까지 반복하며, 이 반복은 16행 printf()의 한 문장으로 구성되므로 중괄호는 필요 없으며, 출력 내용은 저장값인 td[i][j]

17 한 행을 모두 출력한 이후에 다음 줄로 이동하기 위한 출력으로 들여쓰기에 주의

실행결과

```
반목문 for를 이용하여 출력
1 0 0
1 2 3
```

삼차원 배열

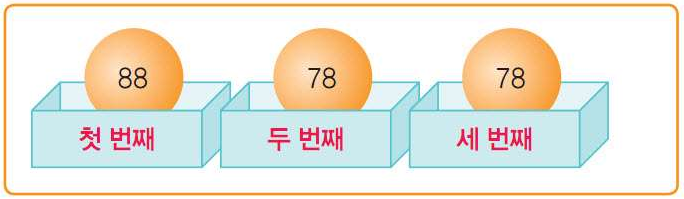
- **삼차원 배열 구조**
  - 다차원 배열을 지원
- **배열 선언**
  - 배열선언 시 대괄호 내부 3개의 크기는 모두 필요

```
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
```

그림 9-21 3차원 배열의 선언과 원소참조

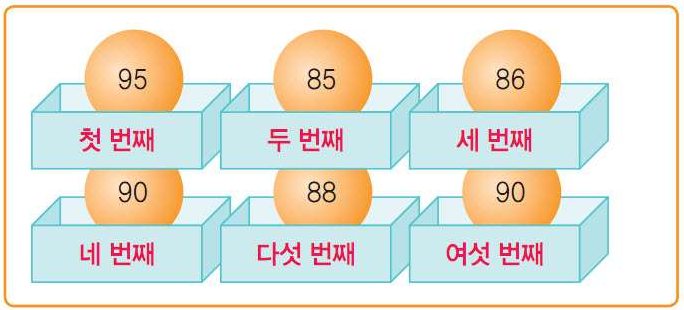
```
int a[3];
```

배열크기 3의 일차원 배열




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

2행 3열의 이차원 배열




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

2 x 2 x 3의 삼차원 배열

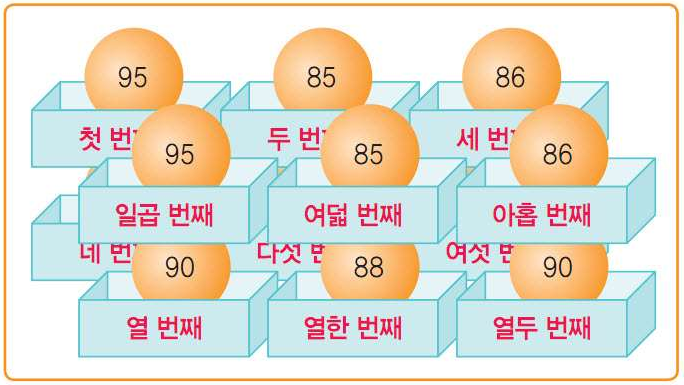


그림 9-20 이차원 배열의 구조

삼차원 배열 초기화

예제 inittwodarray.c

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

초기화 주의점

- 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 } }
};
```

```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;
}
```

그림 9-22 3차원 배열의 예와 초기화 방법

LAB 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.

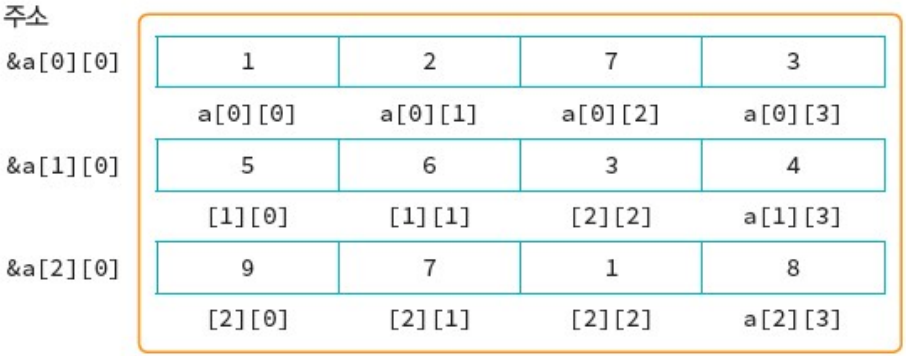


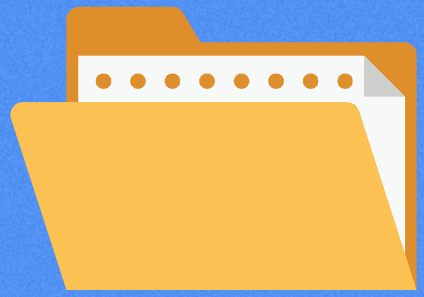
그림 9-23 이차원 배열과 구조

Lab 9-2 arrayprint.c

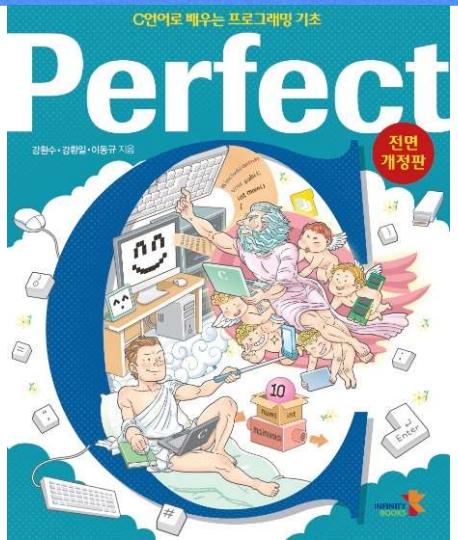
```c
// file: arrayprint.c
#include <stdio.h>

int main()
{
   int a[3][4] = {
      { 1, 2, 7, 3 },   /*  initializers for row indexed by 0 */
      ---------------   /*  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 < ___; j++)
         printf("a[%d][%d] %d %d  ", _____________________________);
      puts("");
   }

   return 0;
}
```

정답

```c
{ 5, 6, 3, 4 },   /*  initializers for row indexed by 1 */
for (int j = 0; j < 4; j++)
    printf("a[%d][%d] %d %d  ", i, j, &a[i][j], a[i][j]);
```



03. 배열과 포인터 관계



일차원 배열과 포인터

- **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]
    - i로 확장하면
      (score + i)는 &score[i]
  - 다시 간접연산자를 사용하면
    - *(score + i)는 score[i]

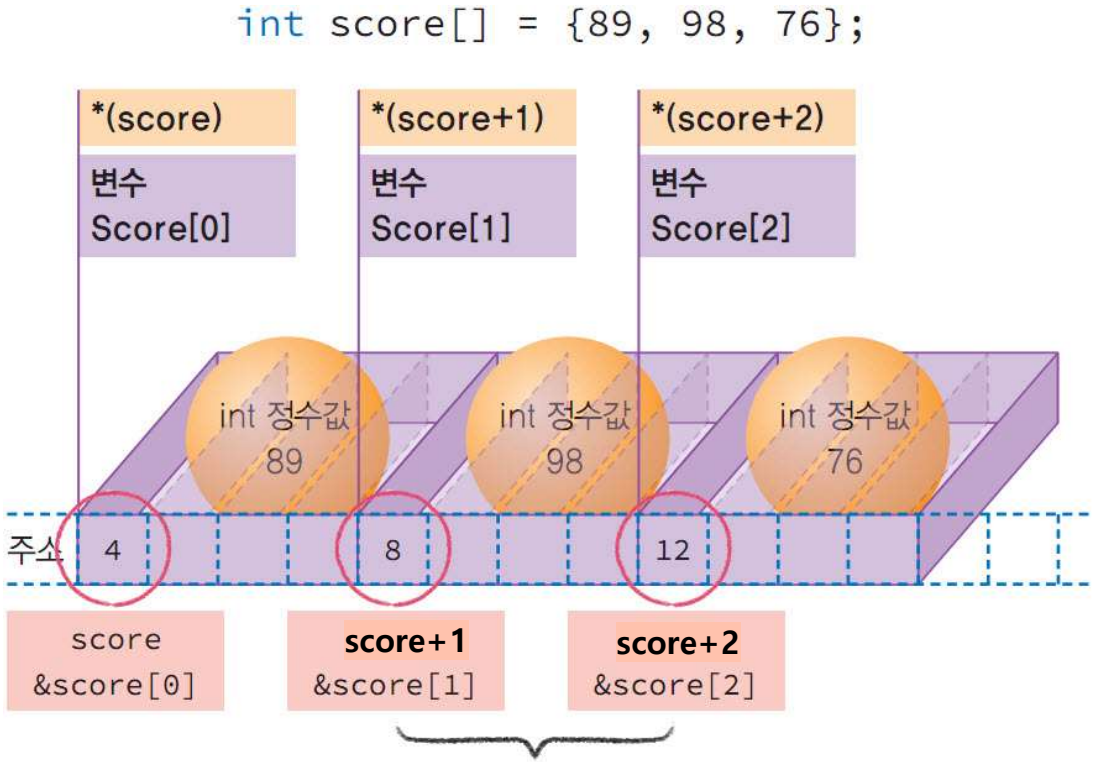


그림 9-24 배열이름을 이용한 배열원소의 참조

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

표 9-1 배열원소의 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

주소값과 참조값 참조

예제 array.c

- 배열이름을 사용한 원소값과 주소값의 참조

포인터 형변환

- 주소값
  - &score[i] == score + i
- 참조값
  - score[i] == *(score + i)

실습예제 9-7 array.c

배열에서 배열이름을 사용한 원소값과 주소값의 참조

```
01  // file: array.c
02  #include <stdio.h>
03  #define SIZE 3
04
05  int main(void)
06  {
07     int score[] = { 89, 98, 76 };
08
09     //배열이름 score는 첫 번째 원소의 주소
10     printf("score: %u, &score[0]: %u\n", score, &score[0]);
11
12     //배열이름 score는 첫 번째 값
13     printf("*score: %d, score[0]: %d\n\n", *score, score[0]);
14
15     printf("첨자    주소      저장값\n");
16     //배열이름 score를 사용한 주소와 원소값 참조
17     for (int i = 0; i < SIZE; i++)
18        printf("%2d %10u %6d\n", i, (score + i), *(score + i));
19
20     return 0;
21  }
```

설명

```
03  배열크기를 매크로 상수 SIZE 3으로 정의
07  int 형 배열 score를 선언하면서 초기화 지정하며, 배열의 크기는 생략되고 초기화 값이 3개이므로
    배열크기는 3으로 자동 지정
10  score, &score[0]는 모두 배열에서 첫 번째 원소의 주소값
13  *score, score[0]는 모두 배열에서 첫 번째 원소의 저장값
15  출력을 위한 제목
17  반복문에서 제어문자 i를 첨자로 사용하며, 0에서 SIZE-1인 2까지 실행
18  반복의 몸체가 printf() 문장 하나로 배열원소를 참조하기 위해 제어문자 i를 첨자로 하여
    (score + i), *(score + i)를 출력, (score + i)는 주소값이며, *(score + i)는
    저장값이 출력
```

실행결과

```
score: 0041F848, &score[0]: 0041F848
*score: 89, score[0]: 89

첨자      주소       저장값
 0   0041F848       89
 1   0041F84C       98
 2   0041F850       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

```
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
```

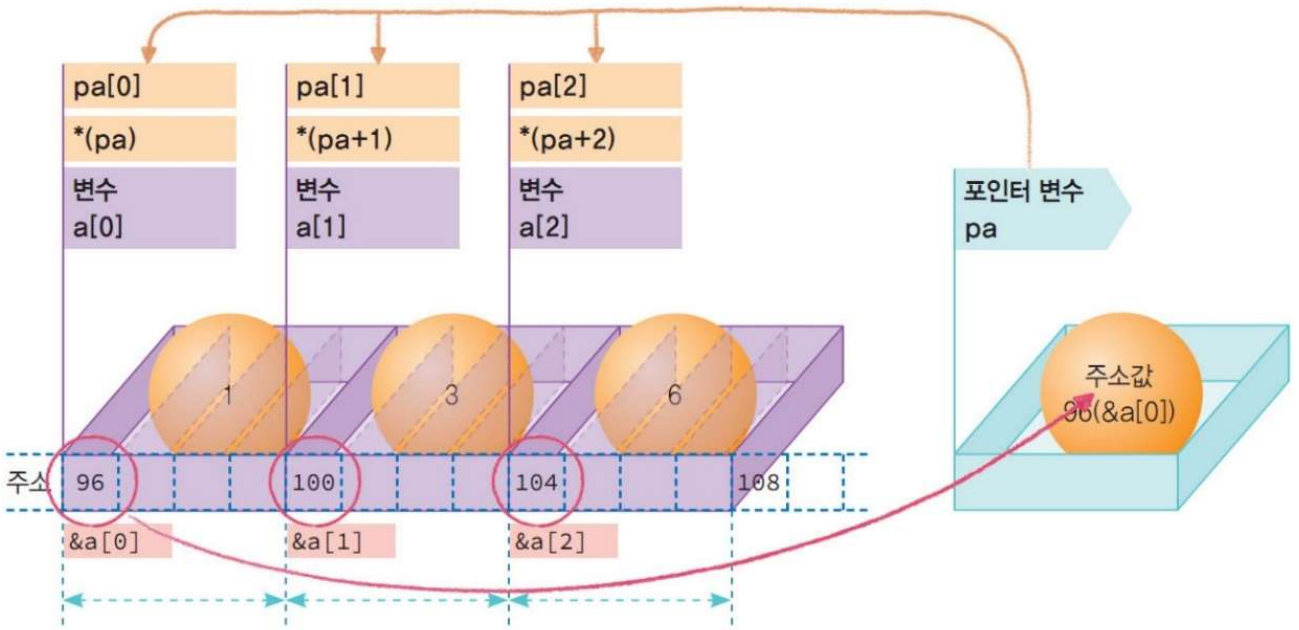


그림 9-25 포인터 변수를 이용한 배열의 참조

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

표 9-2 포인터와 증감연산의 다양한 연산식

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 형변환(type cast)이 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 = (int *) &c[0];
printf("%d %c\n", (int) *pi, (char) *pi); //정수값 65와 문자'A'가 출력
```

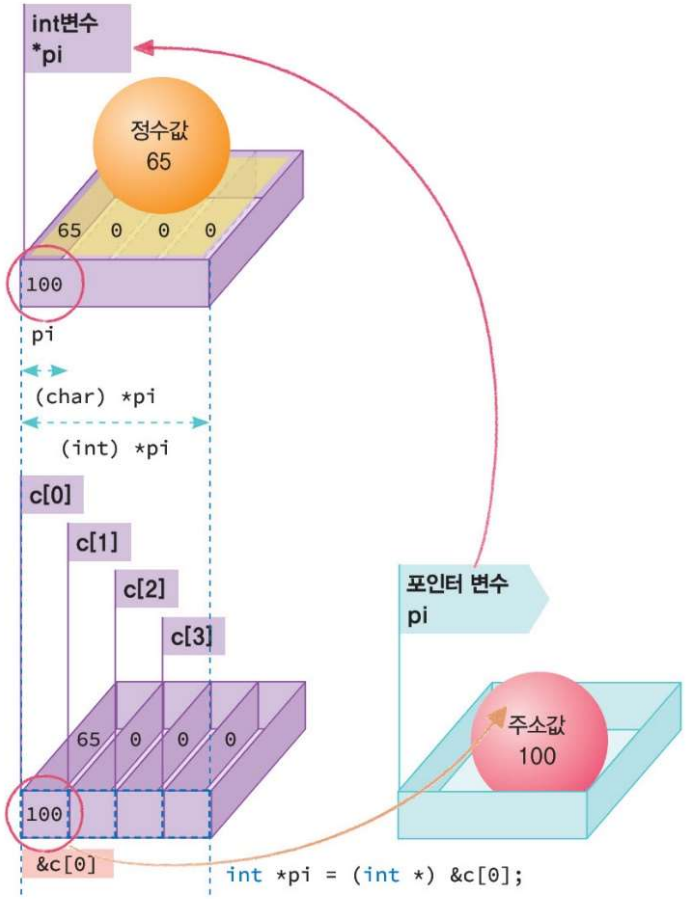


그림 9-27 동일한 메모리의 내용이 포인터의 자료형에 따라 참조 단위가 달라짐

```
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.

그림 9-26 배열 원소의 주소와 포인터 변수 형과의 불일치

포인터 자료형 변환

예제 ptypecast.c

- 포인터의 간접참조
  - 시작주소에서 포인터 자료형 크기만큼 참조

실습예제 9-9 ptypecast.c

포인터 자료형의 변환

```
01  // file: ptypecast.c
02  #include <stdio.h>
03
04  int main(void)
05  {
06     char c[4] = { 'A', '\0', '\0', '\0' }; //문자'A' 코드값: 65
07     //int *pi = &c[0]; //경고 발생
08     int *pi = (int *) &c[0];
09
10     printf("%d %c\n", (int) c[0], c[0]);
11     printf("%d %c\n", *pi, (char) *pi);
12
13     return 0;
14  }
```

설명

06 배열 초기화로 char 배열 선언, 배열 char는 문자 1바이트가 연속적으로 4개이므로 전체는 4바이트

08 int 형 포인터 변수 pi를 선언하면서 배열의 c의 첫 원소의 주소를 저장하는데, (int *)로 변환하여 저장

10 char 형인 c[0]를 int 형으로 변환하여 출력하므로 코드값이 출력, char 형인 c[0]를 출력하면 문자 'A' 출력

11 int *형인 pi를 *pi로 출력하므로 c[0]에서 c[3]에 이르는 4바이트의 정수가 출력되는데, c[1], c[2], c[3]가 모두 0이므로 그대로 코드값인 65가 출력되며, char 형으로 변환하면 1바이트인 c[0]만 출력되는데, %c이므로 문자 'A' 출력

실행결과

```
65 A
65 A
```

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
  - 마찬가지로 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)

```
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;
```

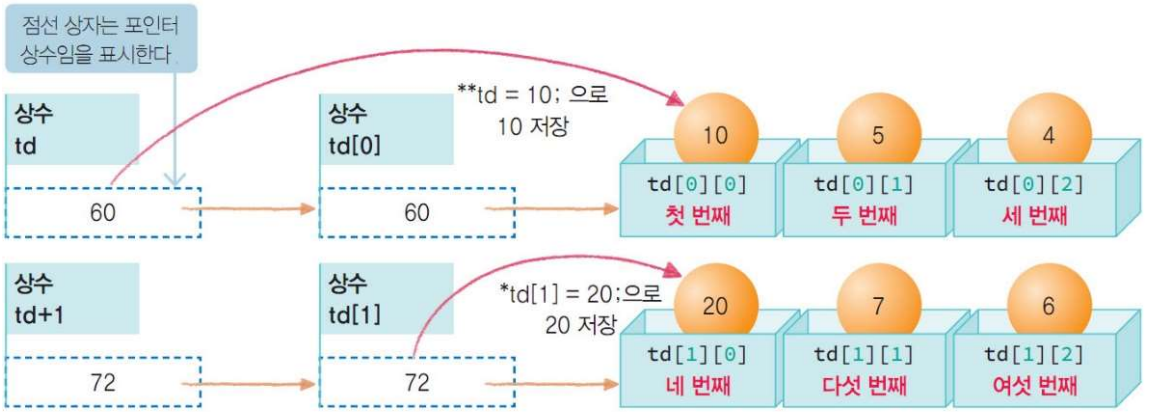


그림 9-28 이차원 배열과 포인터

```
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)

그림 9-29 연산식 *(*td + n)과 *(td[i] + j)으로 배열원소의 참조

이차원 배열 참조

예제 tdaryptr.c

- 배열이름과 행의 대표이름으로 배열원소 참조

실습예제 9-10 tdaryptr.c

배열이름과 행의 대표이름으로 배열원소 참조

```c
// file: 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;
}
```

설명

ROW는 이차원배열 행 크기인 매크로 상수로 양의 정수인 2로 정의, 14행의 for 문의 조건에 사용

COL는 이차원배열 열 크기인 매크로 상수로 양의 정수인 3으로 정의, 16행의 for 문의 조건에 사용

이차원 배열 초기화로 배열 선언

**td는 배열의 첫 원소의 참조 방법

*td[1]는 배열에서 두 번째 행의 첫 원소의 참조 방법

반복문에서 제어문자 i를 첨자로 사용하여 0에서 1까지 반복하며, 이 반복은 내부반복 for와 printf()의 두 문장으로 구성되므로 15행과 21행의 중괄호는 반드시 필요하며, 반복의 초기화 내부 반복에서 사용할 변수 cnt를 선언하고 0으로 저장

내부 반복문에서 제어문자 j를 첨자로 사용하여 0에서 2까지 반복하며, 이 반복은 18행 printf()의 한 문장으로 구성되므로 17과 19줄의 중괄호는 생략될 수 있으며, 출력 내용은 *(*td + cnt), *(td[i] + j), *(*(td + i) + j)으로 배열 td의 모든 원소를 행 별로 한 줄에 출력하는데, *(*td + cnt), *(td[i] + j), *(*(td + i) + j)는 모두 같은 원소를 참조하는 방법

한 행 출력 이후에 다음 줄로 이동하기 위한 출력으로 들여쓰기에 주의

실행결과

```
10 10 10, 5 5 5, 4 4 4,
20 20 20, 7 7 7, 6 6 6,
24, 12, 12
0033F960, 0033F960, 0033F96C
0033F960, 0033F96C
```

변수 double 형에서 int 형 변수 2개 추출

- **실수를 위한 배열크기가 2인 double 형 배열 내부**
  - int 형 자료값을 4개 저장
  - 이 정수가 저장된 공간의 주소와 저장값을 출력
    - 포인터 p를 사용하여 정수 1, 2, 3, 4를 저장
  - 자료형 double은 8바이트
    - 하나의 double 저장 공간에는 4바이트인 2개의 int 형 정수를 저장 가능

Lab 9-3 doubletoint.c

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

int main(void)
{
    double dint[4] = { 0.0 };

    int *p = ______________________;
    p[0] = 1;
    p[1] = 2;
    p[2] = 3;
    p[3] = 4;

    for (int i = 0; i < 4; i++)
        printf("%u %d\n", ____________, ___________________);

    return 0;
}
```

설명

```c
int *p = (int *) dint;
printf("%u %d\n", p + i, *(p + i));
```

```c
double dint[2] = { 0.0 };

int *p = ________________
```

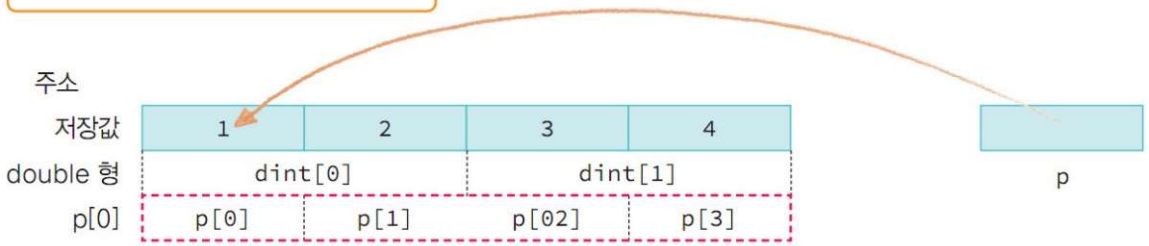
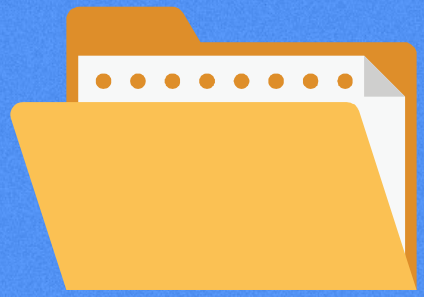
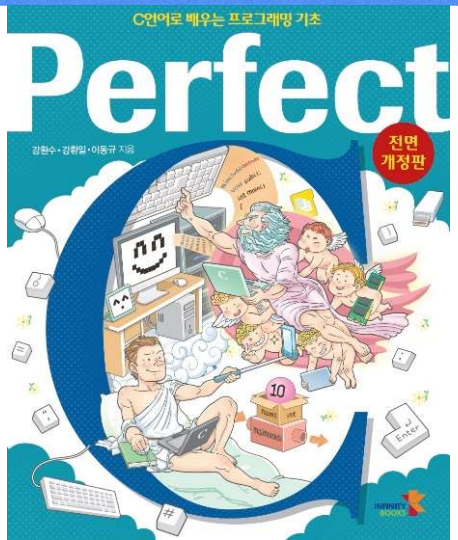


그림 9-30 double 형 2개는 int 형 4개를 저장



04. 포인터 배열과 배열 포인터



포인터 배열(1)

- **포인터 배열(array of pointer) 개요**
  - 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;
```

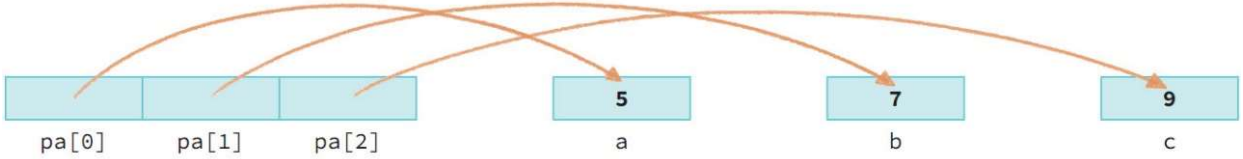


그림 9-31 포인터 배열 pa의 선언과 이해

  - 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};
```

그림 9-32 포인터 배열의 변수 선언 구문

포인터 배열(2)

- **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

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

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

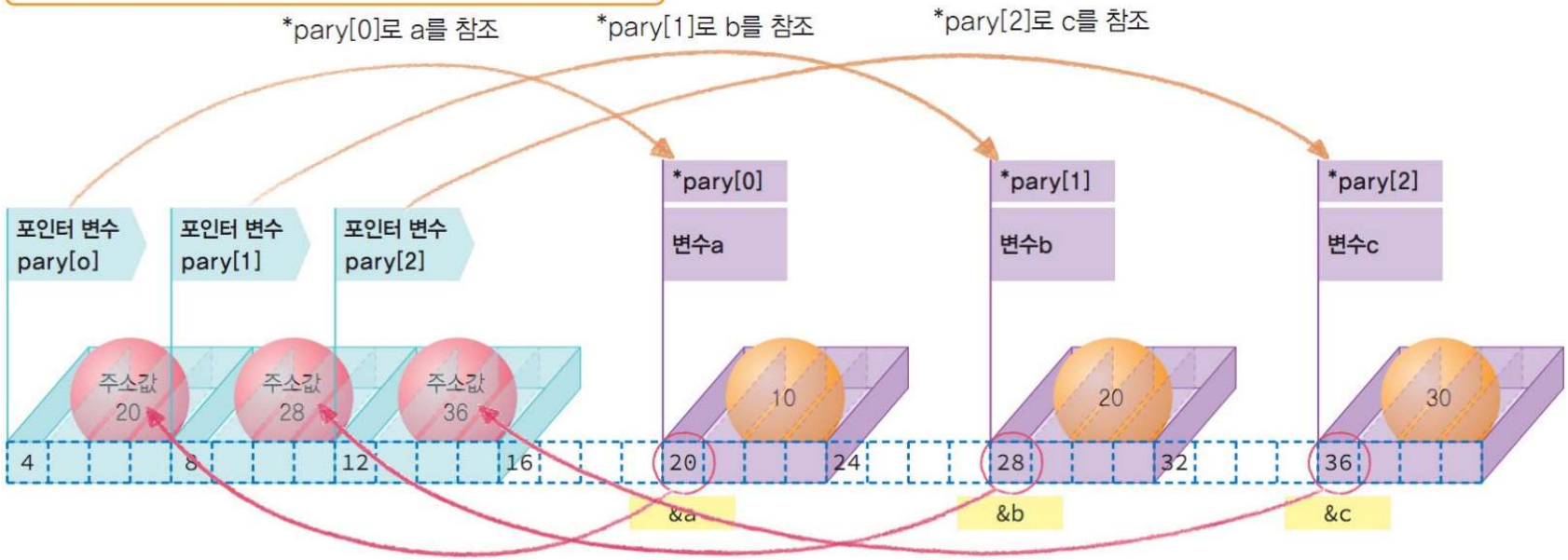


그림 9-33 포인터 배열의 메모리 구조와 역참조

포인터 배열 활용

예제 pointerarray.c

- 포인터 배열 pary를 이용해 표준입력을 받아 다시 원래 변수 a, b, c로 출력

포인터 형변환

- 반복문 내부 scanf()에서 표준입력값이 저장되는 실인자로 pary[i]를 사용하는 것에 주의
  - 일반변수라면 주소연산자 &가 앞에 붙어야 하나
  - pary[i]가 저장할 주소이므로 그대로 사용

실습예제 9-11 pointerarray.c

```c
// file: pointerarray.c
#define _CRT_SECURE_NO_WARNINGS //scanf() 오류를 방지하기 위한 상수 정의
#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;
}
```

pary[i] 자체가 주소값이므로 & 없이 그대로 기술한다.

설명

SIZE는 일차원 배열 크기인 매크로 상수로 양의 정수인 3으로 정의, 16행과 19행의 for 문의 조건에 사용

포인터를 저장하는 일차원 배열 초기화로 배열 선언하면서 모두 NULL 주소를 저장

변수 a, b, c를 각각 선언하면서 각각 10, 20, 30을 저장

포인터 배열 첫 번째 원소 pary[0]에 변수 a의 주소 저장

포인터 배열 두 번째 원소 pary[1]에 변수 b의 주소 저장

포인터 배열 세 번째 원소 pary[2]에 변수 c의 주소 저장

반복문에서 제어문자 i를 첨자로 사용하여 0에서 2까지 반복하며, 이 반복은 printf()의 한 문장으로 구성되므로 중괄호는 필요 없음

*pary[i]는 첨자에 따라 각각 *pary[0], *pary[1], *pary[2]이므로 변수 a, b, c의 저장값을 출력

반복문에서 제어문자 i를 첨자로 사용하여 0에서 2까지 반복하며, 이 반복은 scanf()와 printf()의 두 문장으로 구성되므로 20줄과 23줄의 중괄호가 필요

첨자에 따라 pary[0], pary[1], pary[2]를 scanf()의 인자로 사용하므로, 각각 변수 a, b, c에 표준입력값이 저장

printf() 출력은 계속 a, b, c, 를 출력

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

반복 순서대로 첫 번째 입력값은 a에 다음 순으로 각각 b, c에 저장된다

이차원 배열 포인터(pointer 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

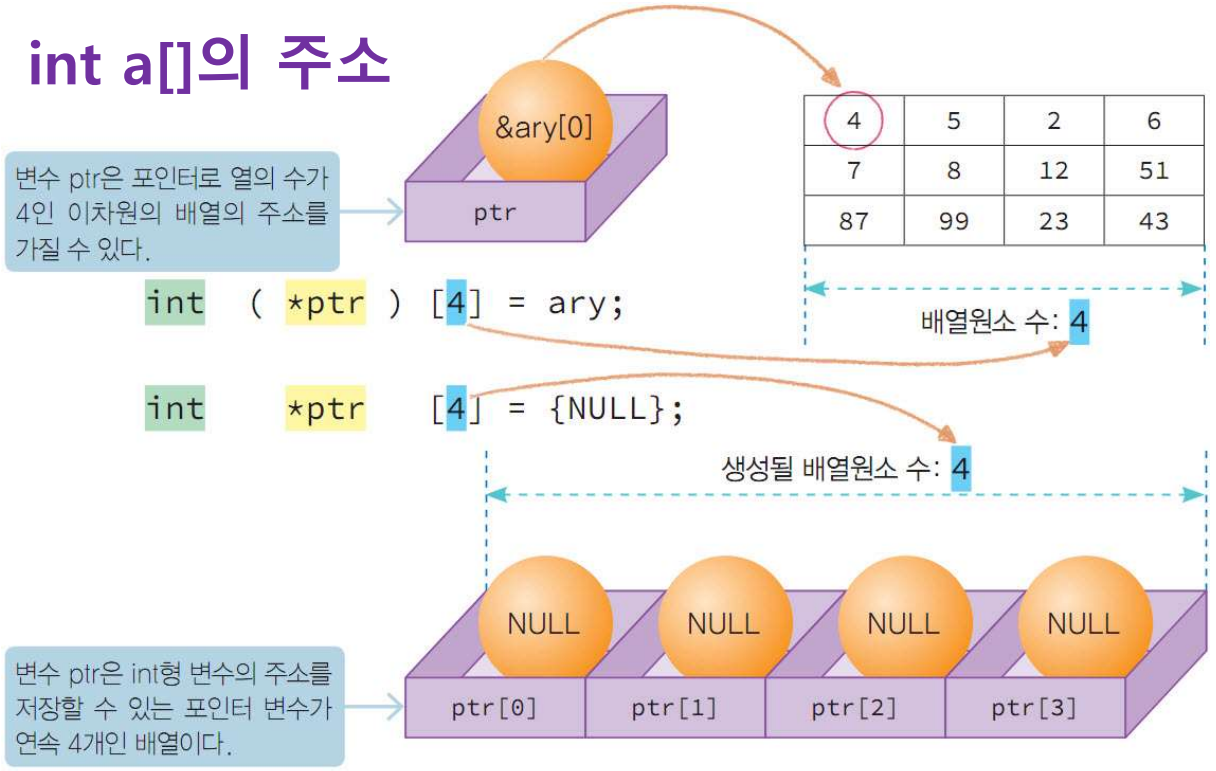


그림 9-35 배열 포인터와 포인터 배열

일차원과 이차원 배열 포인터의 변수 선언

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

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

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

```
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
```

그림 9-34 배열 포인터의 변수 선언 구문

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

```
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
```

그림 9-36 일차원 배열 포인터의 변수선언과 활용

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

```
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));
```

그림 9-37 이차원 배열 포인터의 변수선언과 활용

일차원 배열크기 연산

- **배열크기 계산방법**
  - 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]);
```

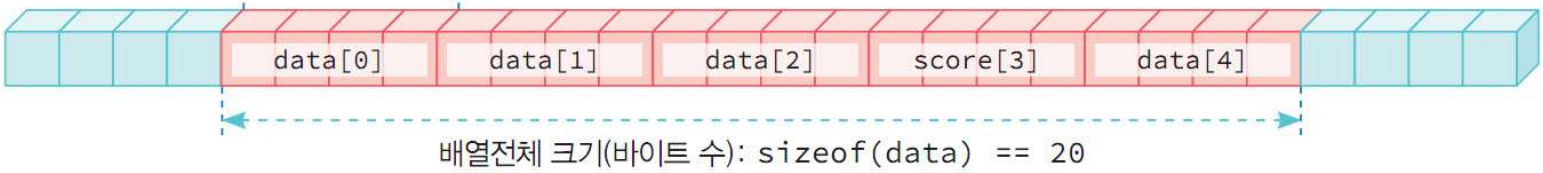


그림 9-38 배열크기인 배열원소의 수 구하기

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

```
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])
```

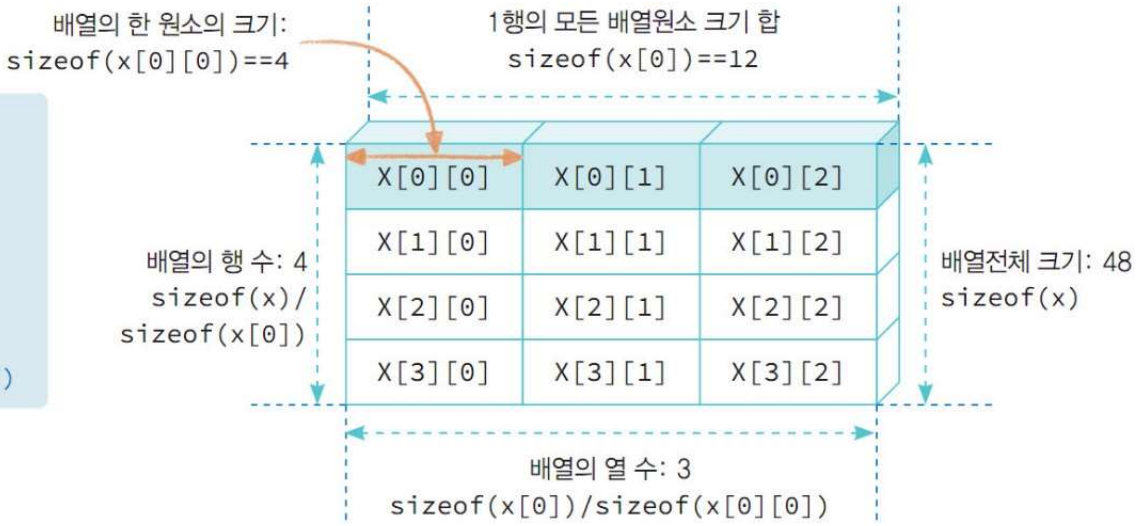


그림 9-39 이차원 배열에서의 행과 열의 수와 전체 배열원소 수 계산

배열에서의 여러 크기

예제 arraysize.c

- 일차원과 이차원 배열에서 배열 전체 및 원소의 크기 출력

실습예제 9-13 arraysize .c

일차원과 이차원 배열에서 배열 전체 및 원소의 크기

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

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

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

   //4 x 3 행렬
   double x[][3] = { { 1, 2, 3 }, { 7, 8, 9 }, { 4, 5, 6 }, { 10, 11, 12 } };

   printf("%d %d %d\n", sizeof(x), sizeof(x[0]), sizeof(x[1]),
      sizeof(x[0][0]));
   int rowsize = sizeof(x) / sizeof(x[0]);
   int colsize = sizeof(x[0]) / sizeof(x[0][0]);
   printf("이차원 배열: 행수 == %d  열수 == %d\n", rowsize, colsize);
   printf("이차원 배열: 전체 원소 수 == %d\n", sizeof(x) / sizeof(x[0][0]));

   return 0;
}
```

설명

06 일차원 배열 data를 선언하면서 초기화 지정

08 sizeof(data)는 배열 전체의 크기인 20이며 sizeof(data[0])는 배열 원소인 data[0] 변수의 크기이므로 4

09 일차원 배열의 원소 수인 배열 크기는 연산식 sizeof(data) / sizeof(data[0]) 으로 계산

12 이차원 배열 x를 선언하면서 초기화 지정

14 sizeof(x)는 배열 전체크기, sizeof(x[0])는 첫 번째 행의 전체크기, sizeof(x[1])는 두 번째 행의 전체크기, sizeof(x[0][0])는 배열을 구성하는 원소 하나의 크기

15 변수 rowsize에는 연산식 sizeof(x) / sizeof(x[0]) 계산으로 이차원 배열의 행 수가 저장

16 변수 colsize에는 연산식 sizeof(x[0]) / sizeof(x[0][0]) 계산으로 이차원 배열의 열 수가 저장

17 변수 rowsize와 colsize 출력

18 이차원 배열의 전체 원소 수는 연산식 sizeof(x) / sizeof(x[0][0]) 으로 출력

실행결과

```
20 4
일차원 배열: 배열 크기 == 5
96 24 24
이차원 배열: 행수 == 4  열수 == 3
이차원 배열: 전체 원소 수 == 12
```

LAB 이차원 배열에서 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

Lab 9-4 twoarysample.c

```c
// 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("%d ", _________________);
    printf("\n\n");

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

    return 0;
}
```

정답

```c
int rowsize = sizeof(abc) / sizeof(abc[0]);
int colsize = sizeof(abc[0]) / sizeof(abc[0][0]);
    printf("%d ", abc[i]);
    printf("%d\n", *p++);
```

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

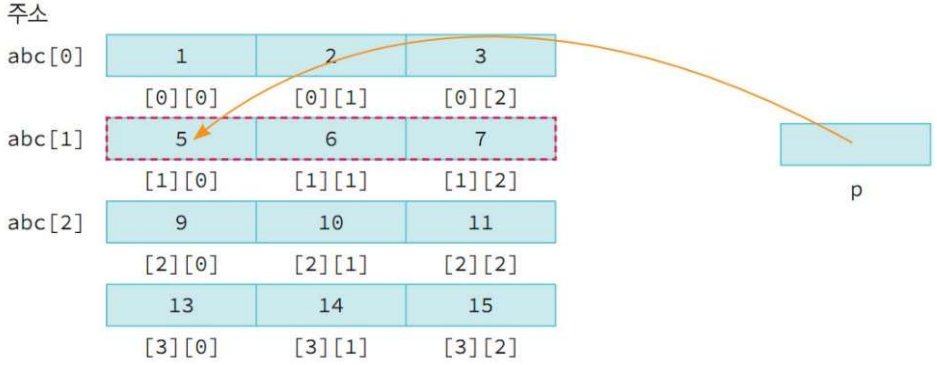


그림 9-40 이차원 배열과 하나의 행

Thank you