Kumoh National Institute of Technology

C Programming

Ch.6-2 다차원 배열

컴퓨터공학과

C프로그래밍

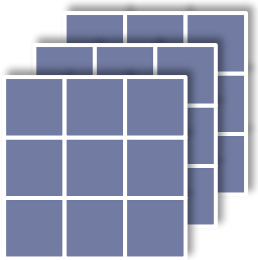
Contents

다차원 배열이란?

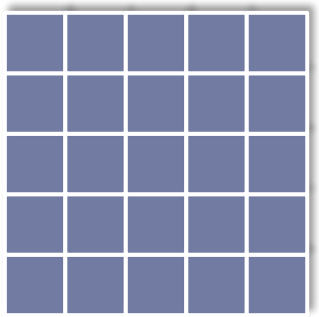
- 다차원 배열 : 2차원 이상의 배열
- 1, 2, 3차원 배열의 선언

```
int arrOneDim[10];          길이가 10인 1차원 int형 배열
int arrTwoDim[5][5];        가로, 세로의 길이가 각각 5인 2차원 int형 배열
int arrThreeDim[3][3][3];   가로, 세로, 높이의 길이가 각각 3인 3차원 int형 배열
```

1차원 배열 arrOneDim



3차원 배열 arrThreeDim



2차원 배열 arrTwoDim

4차원 이상의 배열 선언 및 사용도 가능하지만 이해하기 힘들기 때문에 많이 사용하지 않음
→ 2차원 배열과, 3차원 배열을 능숙하게 사용할 수 있어야 함

2차원 배열의 선언

- 행렬 개념과 유사 : type array_name[행의개수][열의개수]

```
int main(void)
{
    int arr1[3][4];
    int arr2[2][6];

    printf("세로3, 가로4 : %d \n", sizeof(arr1));
    printf("세로2, 가로6 : %d \n", sizeof(arr2));
}
```

세로3, 가로4 : 48 byte
세로2, 가로6 : 48 byte

int arr1[3][4];

int arr2[2][6];

2차원 배열 요소로의 접근 방법

```
int arr[3][3];
```

배열 생성


```
arr[0][0]=1;
```

0 0 접근

배열은 0행 0열부터 시작한다 생각하는 것이 편하다.

[비교] 반복문의 변수

```
arr[0][1]=2;
```

0 1 접근

일반화

```
arr[N-1][M-1]=20;
printf("%d", arr[N-1][M-1]);
```

세로 N, 가로 M의 위치에 값을 저장 및 참조

```
arr[2][1]=5;
```

2 1 접근

2차원 배열 요소로의 접근 예제

- 아파트 : 총 4층 2개 라인 – 각 가구의 인구수 처리

```
int main(void)
{
    int villa[4][2];            //총 4층 1, 2호
    int popu, i, j;

    for (i = 0; i < 4; i++)   // 가구별 거주 인원 입력
    {
        for (j = 0; j < 2; j++)
        {
            printf("%d층 %d호 인구수: ", i + 1, j + 1);
            scanf("%d", &villa[i][j]);
        }
    }

    for (i = 0; i < 4; i++)   // 층별 인구수 출력
    {
        popu = 0;
        popu += villa[i][0];
        popu += villa[i][1];
        printf("%d층 인구수: %d \n", i + 1, popu);
    }
}
```

```
1층 1호 인구수: 2
1층 2호 인구수: 4
2층 1호 인구수: 3
2층 2호 인구수: 5
3층 1호 인구수: 2
3층 2호 인구수: 6
4층 1호 인구수: 4
4층 2호 인구수: 3
1층 인구수: 6
2층 인구수: 8
3층 인구수: 8
4층 인구수: 7
```

2차원 배열은 구조의 특성상 2중 for 문과 잘 어울림

2차원 배열의 메모리 할당 형태

- 1차원 배열의 메모리 할당
  - int arr[6] = { 1, 3, 4, 5, 6, 7 };
- 2차원 배열의 메모리 할당
  - int arr[3][2];
  - 1차원 배열과 동일
    - 다만 접근 방법을 2차원으로 해석

0x1000	1	arr[0]
0x1004	3	arr[1]
0x1008	4	arr[2]
0x100C	5	arr[3]
0x1010	6	arr[4]
0x1014	7	arr[5]

0x1000	?	arr[0][0]
0x1004	?	arr[0][1]
0x1008	?	arr[1][0]
0x100C	?	arr[1][1]
0x1010	?	arr[2][0]
0x1014	?	arr[2][1]

실제로는 시작 주소는 다를 수 있으나 다음 요소들은 연속적으로 할당됨

int arr[2][3];의 메모리 구조는?

2차원 배열의 메모리 할당 예

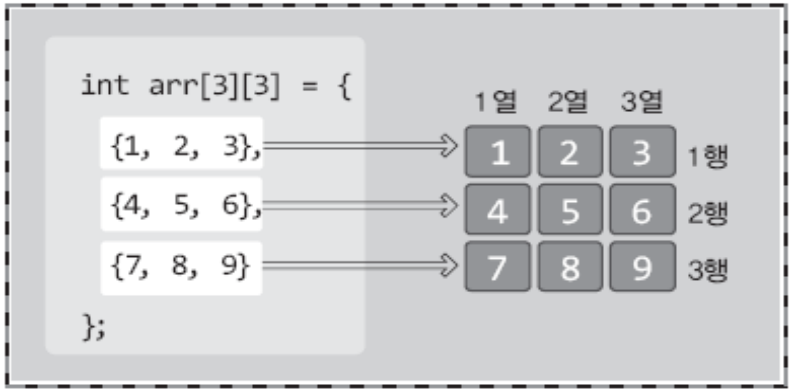
```
int main(void)
{
    int arr[3][2];
    int i, j;

    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            printf("%p %u\n", &arr[i][j] , (unsigned int)&arr[i][j]);
            // 주소값 출력
}
```

```
0078FEFC 7929596
0078FF00 7929600
0078FF04 7929604
0078FF08 7929608
0078FF0C 7929612
0078FF10 7929616
```

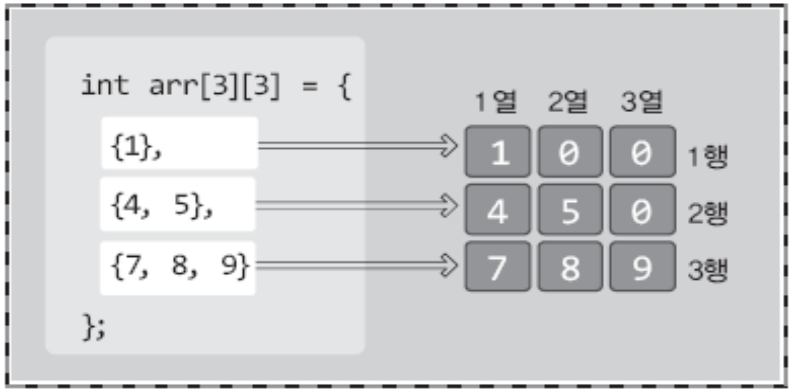
프로그램 개발 시에는 2차원 행렬로 이해하고 사용하면 됨!

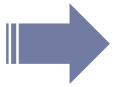
2차원 배열의 선언과 동시에 초기화 (1)

- 초기화 리스트 { } 내에 행 단위로 다시 { } 사용



초기화하지 않으면 지역 변수의 경우 쓰레기값을 가짐



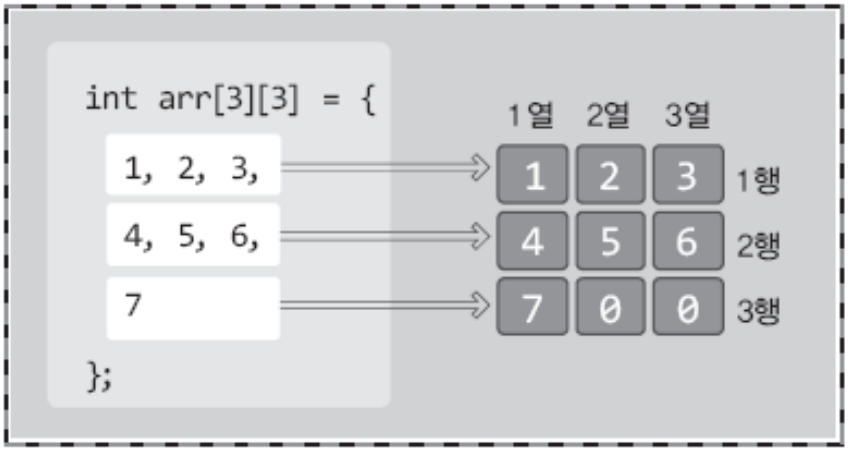


```
int arr[3][3] = {
    {1, 0, 0},
    {4, 5, 0},
    {7, 8, 9}
};
```

초기화 도중 초기화하지 않은 이후 부분은 0으로 초기화

2차원 배열의 선언과 동시에 초기화 (2)

- 각 행에 대하여 별도의 중괄호를 사용하지 않으면
  - 좌 상단부터 행 우선으로 초기화 → 메모리 할당 순서



내부 중괄호 없이 초기화 가능

```
int arr[3][3]={1, 2, 3, 4, 5, 6, 7};
```

초기화 도중 중단하면 나머지는 0

```
int arr[3][3]={1, 2, 3, 4, 5, 6, 7, 0, 0};
```

2차원 배열 선언과 동시에 초기화 (2)

```
int main(void)
{
    int arr1[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
    int arr2[3][3] = { { 1 }, { 4, 5 }, { 7, 8, 9 } };
    int arr3[3][3] = { 1, 2, 3, 4, 5, 6, 7 };

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            printf("%d ", arr1[i][j]);
        printf("\n");
    }
    printf("\n");

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j<3; j++)
            printf("%d ", arr2[i][j]);
        printf("\n");
    }
    printf("\n");

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            printf("%d ", arr3[i][j]);
        printf("\n");
    }
}
```

```
1 2 3
4 5 6
7 8 9

1 0 0
4 5 0
7 8 9

1 2 3
4 5 6
7 0 0
```

배열의 크기를 알려주지 않고 초기화

```
int arr[][] = {1, 2, 3, 4, 5, 6, 7, 8};
```

8 by 1 ??

4 by 2 ??

2 by 4 ??

행과 열 둘 다 생략하면 안됨. 컴파일러를 행, 열의 크기를 결정할 수 없음

```
int arr1[][4] = {1, 2, 3, 4, 5, 6, 7, 8};
int arr2[][2] = {1, 2, 3, 4, 5, 6, 7, 8};
```

컴파일러가 행의 크기를 계산함

```
int arr1[2][4] = {1, 2, 3, 4, 5, 6, 7, 8};
int arr2[4][2] = {1, 2, 3, 4, 5, 6, 7, 8};
```

행의 크기만(배열 이름과 가장 가까운 [] 하나만) 생략할 수 있도록 약속되어 있음

다차원 배열에서 가장 큰 차원의 크기만 생략할 수 있음

3차원 배열의 논리적 구조

- 2차원 배열(행렬)을 높이로 쌓은 구조

```
int main(void)
{
    int arr1[2][3][4];
    double arr2[5][5][5];

    printf("높이2, 세로3, 가로4 int형 배열: %d \n", sizeof(arr1));
    printf("높이5, 세로5, 가로5 double형 배열: %d \n", sizeof(arr2));
}
```

```
높이2, 세로3, 가로4 int형 배열: 96
높이5, 세로5, 가로5 double형 배열: 1000
```

int arr1[2][3][4];
  - 행(세로) 3, 열(가로) 4인 2차원 배열이 2층으로 쌓인 형태

3차원 배열의 선언과 접근 예

- 3개 학급, 각 학급 당 3명, 각각 2개 점수(국어, 수학)

```
int main(void)
{
    int mean = 0, i, j;
    int record[3][3][2] = {
        {
            { 70, 80 },     // A 학급 학생 1의 성적
            { 94, 90 },     // A 학급 학생 2의 성적
            { 70, 85 }      // A 학급 학생 3의 성적
        },
        {
            { 83, 90 },     // B 학급 학생 1의 성적
            { 95, 60 },     // B 학급 학생 2의 성적
            { 90, 82 }      // B 학급 학생 3의 성적
        },
        {
            { 98, 89 },     // C 학급 학생 1의 성적
            { 99, 94 },     // C 학급 학생 2의 성적
            { 91, 87 }      // C 학급 학생 3의 성적
        }
    };
```

```
    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            mean += record[0][i][j];
    printf("A 학급 전체 평균: %g \n", (double)mean / 6);

    mean = 0;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            mean += record[1][i][j];
    printf("B 학급 전체 평균: %g \n", (double)mean / 6);

    mean = 0;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            mean += record[2][i][j];
    printf("C 학급 전체 평균: %g \n", (double)mean / 6);
}
```

```
A 학급 전체 평균: 81.5
B 학급 전체 평균: 83.3333
C 학급 전체 평균: 93
```

이번 장에서 배운 것

- 1차원 배열과 마찬가지로 2차원 배열, 3차원 배열 등 다차원 배열을 만들어 사용할 수 있다.
- 2차원 배열은 행렬의 개념과 유사하며 ary[i][j]와 같이 행index, 열 index를 사용하여 각 요소에 접근할 수 있다. 이때 1차원 배열과 마찬가지로 index는 0부터 시작한다.
- 메모리 할당 형태는 1차원 배열, 2차원 배열 모두 연속적인 주소 상에 할당된다. 2차원 배열의 경우 행우선으로 메모리를 할당하게 된다.
- 3차원 배열은 높이-행-열의 순서로 메모리를 할당하게 된다.
- 4차원 이상의 다차원 배열 또한 사용할 수 있으나 개념적으로 이해가 어려우므로 사용 빈도가 높은 편은 아니다.