

Chapter 18. 다차원 배열과 포인터의 관계



Chapter 18-1. 2차원 배열이름의 포인터 형

1차원 배열이름과 2차원 배열이름의 포인터 형

```
int arr[10];
```

1차원 배열이므로 arr은 int형 포인터 (int *)

```
int * parr[20];
```

1차원 배열이므로 parr은 int형 이중 포인터 (int **)

```
int arr2d[3][4];
```

int형 1차원 배열도, int 포인터 형 1차원 배열도 아니므로 arr2d는 int형 포인터 형도, int형 이중 포인터 형도 아니다. 2차원 배열이름의 포인터 형을 결정짓는 방법은 별도로 존재한다.



```
int main(void)
{
    int arr2d[3][3];
    printf("%d \n", arr2d);
    printf("%d \n", arr2d[0]);
    printf("%d \n\n", &arr2d[0][0]);

    printf("%d \n", arr2d[1]);
    printf("%d \n\n", &arr2d[1][0]);

    printf("%d \n", arr2d[2]);
    printf("%d \n\n", &arr2d[2][0]);

    printf("sizeof(arr2d): %d \n", sizeof(arr2d));
    printf("sizeof(arr2d[0]): %d \n", sizeof(arr2d[0]));
    printf("sizeof(arr2d[1]): %d \n", sizeof(arr2d[1]));
    printf("sizeof(arr2d[2]): %d \n", sizeof(arr2d[2]));
    return 0;
}
```

2차원 배열이름이 가리키는 것들은?

2차원 배열이름의 포인터 형을 결정지으려면 우선 2차원 배열이름이 가리키는 대상이 무엇인지 알아야 한다. 그런데 1차원 배열과 달리 이것만으로 포인터 형이 결정되지 않는다.

```
int arr2d[3][3];
```



배열이름 arr2d가 가리키는 것은 인덱스 기준으로 [0][0]에 위치한 첫 번째 요소



2차원 배열의 경우 arr2d[0], arr2d[1], arr2d[2]도 의미를 지닌다. 각각 1행, 2행, 3행의 첫 번째 요소를 가리키는 주소 값의 의미를 지닌다.

그럼 arr2d와 arr2d[0]는 같은 것인가?

```
int main(void)
{
    int arr2d[3][3];
    printf("%d \n", arr2d);
    printf("%d \n", arr2d[0]);
    printf("%d \n\n", &arr2d[0][0]);

    printf("%d \n", arr2d[1]);
    printf("%d \n\n", &arr2d[1][0]);

    printf("%d \n", arr2d[2]);
    printf("%d \n\n", &arr2d[2][0]);

    printf("sizeof(arr2d): %d \n", sizeof(arr2d));
    printf("sizeof(arr2d[0]): %d \n", sizeof(arr2d[0]));
    printf("sizeof(arr2d[1]): %d \n", sizeof(arr2d[1]));
    printf("sizeof(arr2d[2]): %d \n", sizeof(arr2d[2]));
    return 0;
}
```

arr2d는 2차원 배열 전체를 의미한다.. 반면 arr2d[0]는 2차원 배열의 첫 번째 행을 의미한다.

실행결과

```
4585464
4585464
4585464
4585476
4585476
4585488
4585488
sizeof(arr2d): 36
sizeof(arr2d[0]): 12
sizeof(arr2d[1]): 12
sizeof(arr2d[2]): 12
```

배열이름 arr2d를 대상으로 sizeof 연산을 하는 경우 배열 전체의 크기를 반환

arr2d[0], arr2d[1], arr2d[2]를 대상으로 sizeof 연산을 하는 경우 각 행의 크기를 반환

배열이름 기반의 포인터 연산

```
int iarr[3];      // iarr은 int형 포인터  
double darr[7];  // darr은 double형 포인터
```

```
printf("%p", iarr+1);  
printf("%p", darr+1);
```

iarr은 **int형 포인터이기 때문에** +1의 결과로 sizeof(int)의 크기만큼 값이 증가한다.

darr은 **double형 포인터이기 때문에** +1의 결과로 sizeof(double)의 크기만큼 값이 증가한다.

이렇듯 포인터 연산의 결과는 포인터 형에 의존적이다. 따라서 2차원 배열이름의 포인터 형을 결정짓기 위한 힌트는 포인터 연산의 결과를 주목하면 얻을 수 있다.



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

    printf("arr1: %p \n", arr1);
    printf("arr1+1: %p \n", arr1+1);
    printf("arr1+2: %p \n\n", arr1+2);

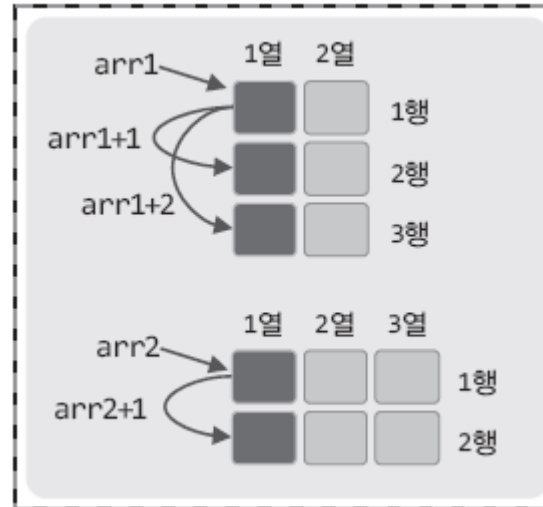
    printf("arr2: %p \n", arr2);
    printf("arr2+1: %p \n", arr2+1);
    return 0;
}
```


2차원 배열이름 대상의 포인터 연산 결과

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

    printf("arr1: %p \n", arr1);
    printf("arr1+1: %p \n", arr1+1);
    printf("arr1+2: %p \n\n", arr1+2);

    printf("arr2: %p \n", arr2);
    printf("arr2+1: %p \n", arr2+1);
    return 0;
}
```



2차원 배열이름을 대상으로 값을 1씩 증가 및 감소하는 경우 그 결과는 각 행의 첫 번째 요소의 주소 값이다.

```
arr1: 004BFBE0
arr1+1: 004BFBE8
arr1+2: 004BFBF0

arr2: 004BFBC0
arr2+1: 004BFBC8
```

`arr1`과 `arr2`는 둘 다 `int`형 2차원 배열이다. 그러나 **가로의 길이가 다르기 때문에** 포인터 연산결과로 증가 및 감소하는 값의 크기에는 차이가 있다. 즉, **`arr1`과 `arr2`는 의 포인터 형은 일치하지 않는다.**

이렇듯 2차원 배열이름의 포인터 형은 배열의 가로길이에 따라서도 나뉜다. 그리고 이러한 특징 때문에 2차원 배열이름의 포인터 형 결정이 쉽지 않은 것이다.

실행결과

최종결론! 2차원 배열이름의 포인터 형 1

1. 2차원 배열이름의 포인터 형을 결정짓는 두 가지 요소!

1. 가리키는 대상은 무엇인가?
2. 배열이름(포인터)를 대상으로 값을 1 증가 및 감소 시 실제로는 얼마가 증가 및 감소하는가?

2. 왜 다른가?

1차원 배열이름의 자료형은 1차원 배열이름이 가리키는 대상만으로 결정이 되는데 2차원 배열이름의 자료형은 그렇지 않은 이유는?

3. 포인터 형을 통한 메모리의 접근과 주소 값의 증가

1차원 배열의 경우에는 배열이름이 가리키는 대상을 기준으로 메모리의 접근방법과 포인터 연산시의 증가 및 감소의 크기가 결정되었다. 그러나 2차원 배열에서는 위의 두 가지 정보가 모두 존재해야 이 둘을 결정지을 수 있다.



최종결론! 2차원 배열이름의 포인터 형 2

int arr[3][4]의 포인터 형은?

어색하지만 이것이 arr의 포인터 형을 설명하는 최선의 방법이다.
int형 포인터, double형 포인터와 같이 딱 떨어지는 명칭이 존재하지 않는다.

1. 가리키는 대상 int형 변수
2. 포인터 연산의 결과 sizeof(int)×4의 크기단위로 값이 증가 및 감소



이러한 유형의 포인터 변수 ptr의 선언

```
int (*ptr) [4];
```

2차원 배열을 가리키는 포인터
변수이므로 배열 포인터 변수
라 한다.

```
int (*ptr) [4]
```

ptr은 포인터!

```
int (*ptr) [4]
```

int형 변수를 가리키는 포인터 !

```
int (*ptr) [4]
```

포인터 연산 시 4칸씩 건너뛰는 포인터!

2차원 배열 이름의 포인터 형 결정짓는 연습

case 1

```
char (*arr1)[4];  
double (*arr2)[7];
```



“arr1은 char형 변수를 가리키면서, 포인터 연산 시 $\text{sizeof(char)} \times 4$ 의 크기단위로 값이 증가 및 감소하는 포인터 변수”

“arr2는 double형 변수를 가리키면서, 포인터 연산 시 $\text{sizeof(double)} \times 7$ 의 크기단위로 값이 증가 및 감소하는 포인터 변수”

case 2

“int형 변수를 가리키면서, 포인터 연산 시 $\text{sizeof(int)} \times 2$ 의 크기단위로 값이 증가 및 감소하는 포인터 변수 ptr1”

“float형 변수를 가리키면서, 포인터 연산 시 $\text{sizeof(float)} \times 5$ 의 크기단위로 값이 증가 및 감소하는 포인터 변수 ptr2”



```
int (*ptr1)[2];  
float (*ptr2)[5];
```

실습 3

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

    int (*ptr)[2];
    int i;

    ptr=arr1;
    printf("*** Show 2,2 arr1 **\n");
    for(i=0; i<2; i++)
        printf("%d %d \n", ptr[i][0], ptr[i][1]);

    ptr=arr2;
    printf("*** Show 3,2 arr2 **\n");
    for(i=0; i<3; i++)
        printf("%d %d \n", ptr[i][0], ptr[i][1]);

    ptr=arr3;
    printf("*** Show 4,2 arr3 **\n");
    for(i=0; i<4; i++)
        printf("%d %d \n", ptr[i][0], ptr[i][1]);

    return 0;
}
```

2차원 배열이름의 포인터 관련 예제

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

    int (*ptr)[2];
    int i;

    ptr=arr1;
    printf("*** Show 2,2 arr1 **\n");
    for(i=0; i<2; i++)
        printf("%d %d \n", ptr[i][0], ptr[i][1]);

    ptr=arr2;
    printf("*** Show 3,2 arr2 **\n");
    for(i=0; i<3; i++)
        printf("%d %d \n", ptr[i][0], ptr[i][1]);

    ptr=arr3;
    printf("*** Show 4,2 arr3 **\n");
    for(i=0; i<4; i++)
        printf("%d %d \n", ptr[i][0], ptr[i][1]);

    return 0;
}
```

arr1, arr2, arr3는 둘 다 int형 2차원 배열이면서 가로 길이가 같으므로 포인터 형이 동일하다.

실행결과

```
** Show 2,2 arr1 **
1 2
3 4
** Show 3,2 arr2 **
1 2
3 4
5 6
** Show 4,2 arr3 **
1 2
3 4
5 6
7 8
```



Chapter 18-2. 2차원 배열 이름의 특성과 주의사항

실습 4

```
int main(void)
{
    int num1=10, num2=20, num3=30, num4=40;
    int arr2d[2][4]={1, 2, 3, 4, 5, 6, 7, 8};
    int i, j;

    int * whoA[4]={&num1, &num2, &num3, &num4};    // 포인터 배열
    int (*whoB)[4]=arr2d;    // 배열 포인터

    printf("%d %d %d %d \n", *whoA[0], *whoA[1], *whoA[2], *whoA[3]);
    for(i=0; i<2; i++)
    {
        for(j=0; j<4; j++)
            printf("%d ", whoB[i][j]);
        printf("\n");
    }
    return 0;
}
```



'배열 포인터'와 '포인터 배열'을 혼동하지 말자

```
int * whoA [4];    // 포인터 배열
int (*whoB) [4];   // 배열 포인터
```

포인터 배열

포인터 변수로 이루어진 배열

배열 포인터

배열을 가리킬 수 있는 포인터 변수

```
int main(void)
{
    int num1=10, num2=20, num3=30, num4=40;
    int arr2d[2][4]={1, 2, 3, 4, 5, 6, 7, 8};
    int i, j;

    int * whoA[4]={&num1, &num2, &num3, &num4};    // 포인터 배열
    int (*whoB)[4]=arr2d;    // 배열 포인터

    printf("%d %d %d %d \n", *whoA[0], *whoA[1], *whoA[2], *whoA[3]);
    for(i=0; i<2; i++)
    {
        for(j=0; j<4; j++)
            printf("%d ", whoB[i][j]);
        printf("\n");
    }
    return 0;
}
```

실행결과

```
10 20 30 40
1 2 3 4
5 6 7 8
```

2차원 배열을 함수의 인자로 전달하기

```
int main(void)
{
    int arr1[2][7];
    double arr2[4][5];
    SimpleFunc(arr1, arr2);
    . . .
}
```

매개변수의 선언 위치에서만 동일한 선언으로
간주된다.

```
void SimpleFunc( int (*parr1)[7], double (*parr2)[5] ) { . . . }
```



동일한 선언



동일한 선언

```
void SimpleFunc( int parr1[][7], double parr2[][5] ) { . . . }
```

실습 5

```
void ShowArr2DStyle(int (*arr)[4], int column)
{
    // 배열요소 전체출력
    int i, j;
    for(i=0; i<column; i++)
    {
        for(j=0; j<4; j++)
            printf("%d ", arr[i][j]);
        printf("\n");
    }
    printf("\n");
}
```

```
int Sum2DArr(int arr[][4], int column)
{
    // 배열요소의 합 반환
    int i, j, sum=0;
    for(i=0; i<column; i++)
        for(j=0; j<4; j++)
            sum += arr[i][j];
    return sum;
}
```

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

    ShowArr2DStyle(arr1, sizeof(arr1)/sizeof(arr1[0]));
    ShowArr2DStyle(arr2, sizeof(arr2)/sizeof(arr2[0]));
    printf("arr1의 합: %d \n", Sum2DArr(arr1, sizeof(arr1)/sizeof(arr1[0])));
    printf("arr2의 합: %d \n", Sum2DArr(arr2, sizeof(arr2)/sizeof(arr2[0])));
    return 0;
}
```

2차원 배열을 함수의 인자로 전달하는 예제

```
void ShowArr2DStyle(int (*arr)[4], int column)
{
    // 배열요소 전체출력
    int i, j;
    for(i=0; i<column; i++)
    {
        for(j=0; j<4; j++)
            printf("%d ", arr[i][j]);
        printf("\n");
    }
    printf("\n");
}
```

```
int Sum2DArr(int arr[][4], int column)
{
    // 배열요소의 합 반환
    int i, j, sum=0;
    for(i=0; i<column; i++)
        for(j=0; j<4; j++)
            sum += arr[i][j];
    return sum;
}
```

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

```
1 1 1 1
3 3 3 3
5 5 5 5
```

```
arr1의 합: 36
arr2의 합: 36
```

실행결과

정의된 두 함수의 인자로 전달되는 2차원 배열의 **가로길이는 결정되어 있다.**

반면, **세로 길이 정보는 결정되어 있지 않고** 두 번째 인자를 통해서 추가로 전달하고 있다. 이점에 주목하자!

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

    ShowArr2DStyle(arr1, sizeof(arr1)/sizeof(arr1[0]));
    ShowArr2DStyle(arr2, sizeof(arr2)/sizeof(arr2[0]));
    printf("arr1의 합: %d \n", Sum2DArr(arr1, sizeof(arr1)/sizeof(arr1[0])));
    printf("arr2의 합: %d \n", Sum2DArr(arr2, sizeof(arr2)/sizeof(arr2[0])));
    return 0;
}
```

배열의 세로길이 계산방식

```
int main(void)
{
    int a[3][2]={1, 2}, {3, 4}, {5, 6}};
    printf("a[0]: %p \n", a[0]);
    printf("*(a+0): %p \n", *(a+0));
    printf("a[1]: %p \n", a[1]);
    printf("*(a+1): %p \n", *(a+1));
    printf("a[2]: %p \n", a[2]);
    printf("*(a+2): %p \n", *(a+2));
    printf("%d, %d \n", a[2][1], (*(a+2))[1]);
    printf("%d, %d \n", a[2][1], *(a[2]+1));
    printf("%d, %d \n", a[2][1], *(*a+2)+1));
    return 0;
}
```

2차원 배열에서도 arr[i]와 *(arr+i)는 같다.

`arr[i] == *(arr+i)`

Ch13에서 1차원 배열과 포인터 변수를 대상으로 내린 결론! 2차원 배열에서도 그대로 적용이 된다!

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

```
arr[2][1]=4;  
(*(arr+2))[1]=4;  
*(arr[2]+1)=4;  
*(*(arr+2)+1)=4;
```

위에 선언된 배열 arr을 대상으로 인덱스 기준 [2][1] 번째 요소에 4를 저장하는, 모두 동일한 결과를 보이는 문장이다.

```
a[0]: 001AFDC8  
*(a+0): 001AFDC8  
a[1]: 001AFDD0  
*(a+1): 001AFDD0  
a[2]: 001AFDD8  
*(a+2): 001AFDD8  
  
6, 6  
6, 6  
6, 6
```

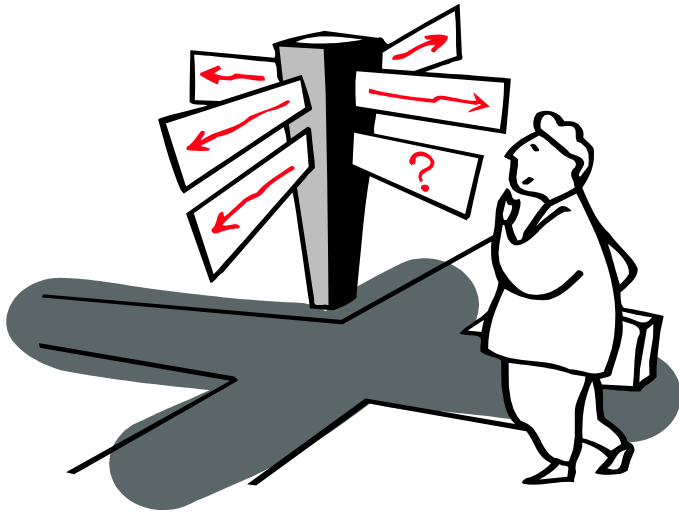
실행결과

```
int main(void)  
{  
    int a[3][2]={ {1, 2}, {3, 4}, {5, 6} };  
    printf("a[0]: %p \n", a[0]);  
    printf("*(a+0): %p \n", *(a+0));  
    printf("a[1]: %p \n", a[1]);  
    printf("*(a+1): %p \n", *(a+1));  
    printf("a[2]: %p \n", a[2]);  
    printf("*(a+2): %p \n", *(a+2));  
    printf("%d, %d \n", a[2][1], (*(a+2))[1]);  
    printf("%d, %d \n", a[2][1], *(a[2]+1));  
    printf("%d, %d \n", a[2][1], *(*a+2)+1));  
    return 0;  
}
```

아래의 코드를 보자(3차원 배열이라고 겁을 먹을 필요는 없다. 이 문제는 3차원 배열의 특성을 몰라도 해결할 수 있는 문제다). 이 예제에서는 인덱스 [1][0][1]의 배열요소를 출력하고 있다. 이 때 사용된 `arr[1][0][1]`을 대신할 수 있는 문장을 5개 이상 제시해보자.

```
int main(void)
{
    int arr[2][2][2]={1, 2, 3, 4, 5, 6, 7, 8};
    printf("%d \n", arr[1][0][1]);
    return 0;
}
```





Chapter 18이 끝났습니다. 질문 있으신
지요?