다차원 배열과 포인터

- 2차원 배열 이름 -

성공회대학교 IT융합자율학부
소프트웨어공학전공
홍 성 준

1차원 배열 이름의 포인터 형

◎ 1차원 배열 이름의 포인터 형

- 1차원 배열 이름은 배열의 첫번째 요소를 가리키는 포인터

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

  - arr은 int 형 포인터 (int *)

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

  - parr은 int* 형 포인터 (int 형 더블 포인터, int **)

◎ 2차원 배열 이름의 포인터 형은?

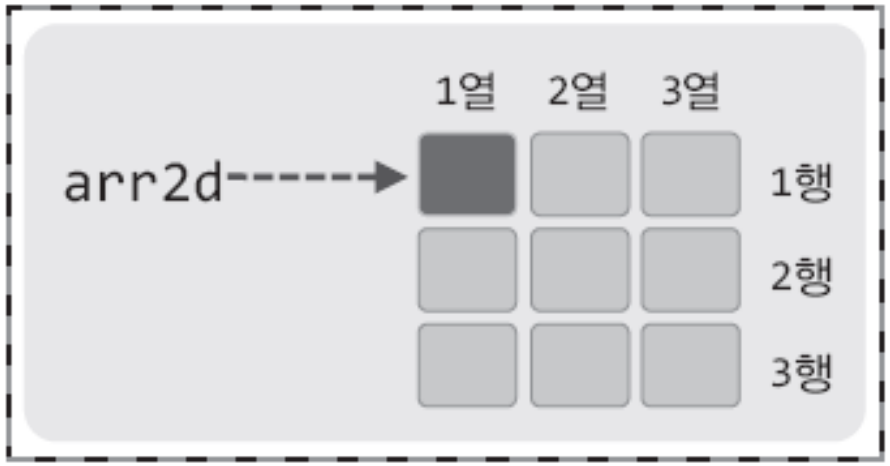
```
int arr2d[3][3];
```

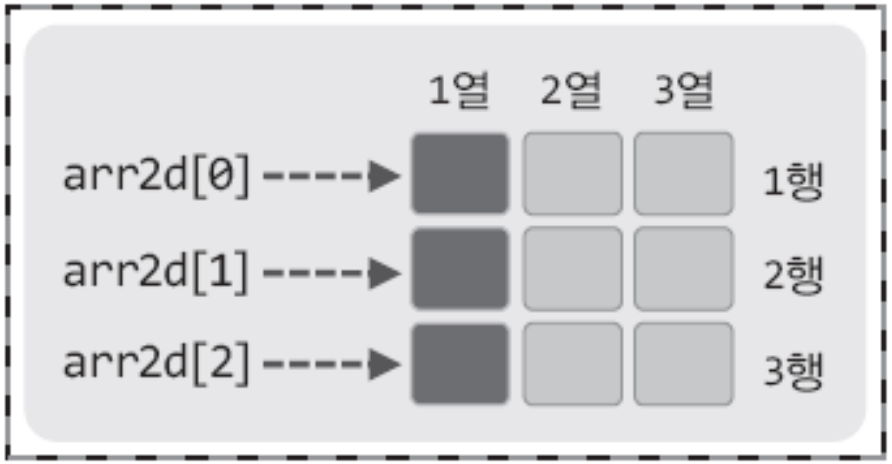
2차원 배열 이름

◎ 2차원 배열 이름

- 2차원 배열 이름은 인덱스 기준 [0][0]에 위치한 2차원 배열의 첫번째 요소를 가리키는 포인터
- 2차원 배열의 경우 배열 이름과 행 인덱스를 사용하여 각 행의 첫번째 요소를 가리킬 수 있음

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





- 2차원 배열의 이름을 arr2d라고 할 때, arr2d 와 arr2d[0]는 같은 배열의 요소를 나타냄

2차원 배열 이름

◎ 2DArrayAddress.c

```
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;
}
```

```
4585464
4585464
4585464

4585476
4585476

4585488
4585488

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

- sizeof(arr2d)는 배열 전체 크기를 반환하고, sizeof(arr2d[i])는 각 행의 크기를 반환

배열 이름 기반 포인터 연산

◎ 1차원 배열 이름 기반의 포인터 연산

```
int iarr[3];          // iarr은 int형 포인터
double darr[7];       // darr은 double형 포인터
printf("%p", iarr+1);
printf("%p", darr+1);
```

- iarr은 int 형 포인터이기 때문에 +1의 결과로 sizeof(int)만큼 값이 증가
- darr은 double 형 포인터이기 때문에 +1의 결과로 sizeof(double)만큼 값이 증가

배열 이름 기반 포인터 연산

◎ 2DArrayAddress.c

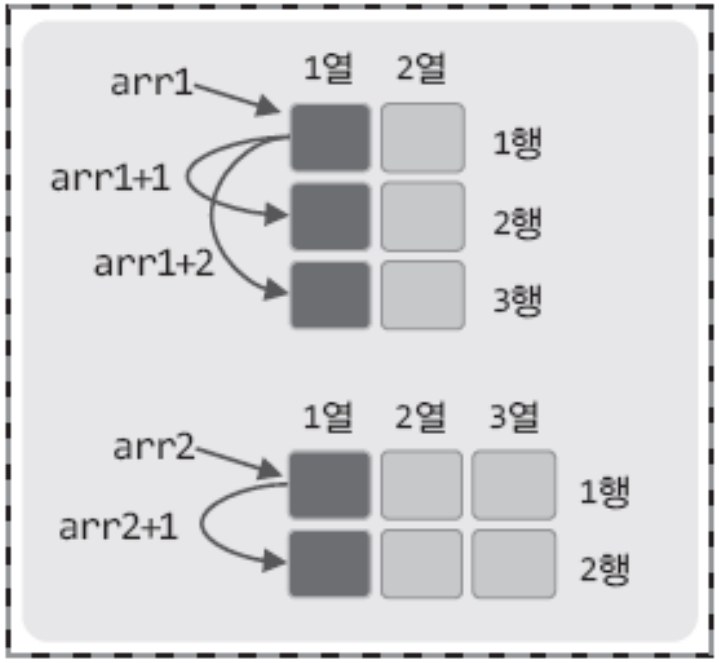
```
int main(void)
{
    int arr1[3][2];
    int arr2[2][3];

    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;
}
```

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

arr2: 004BFBC0
arr2+1: 004BFBCC
```



- 2차원 배열 이름을 대상으로 증감 연산을 하면 결과는 각 행의 첫번째 요소의 주소 값
- 즉, 2차원 배열 이름의 포인터 형은 배열의 가로 길이에 따라 달라짐
- 실제 arr1과 arr2 모두 2차원 배열이지만, 포인터 증감 연산 결과에는 차이가 있음

2차원 배열 이름의 포인터 형

◎ 2차원 배열 이름의 포인터 형을 결정 짓는 요소

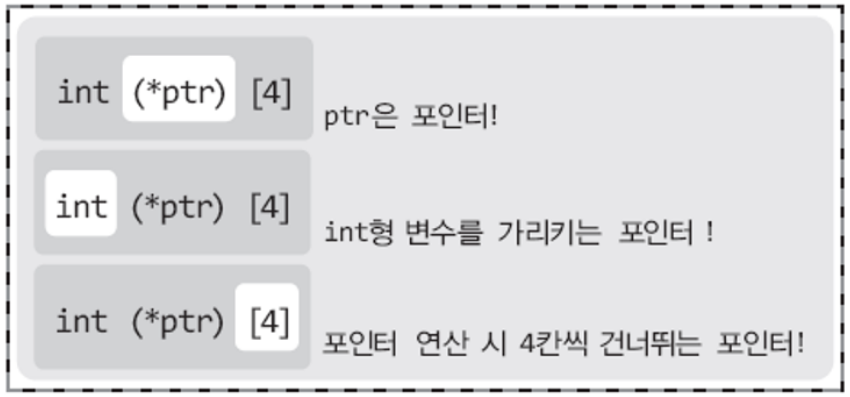
- 가리키는 대상이 무엇인가?
- 배열 이름을 대상으로 값을 증감할 때 주소값은 실제 얼마나 증감하는가?

2차원 배열 이름의 포인터 형

◎ int arr2d[3][4] 의 포인터 형은?

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

int (*ptr) [4];



- ptr은 int형 변수를 가리키면서, 포인터 연산 시 sizeof(int) * 4 크기 단위로 값이 증감하는 포인터 변수
- 이와 같이 배열을 가리키는 포인터 변수를 '배열 포인터 변수'라고 함

2차원 배열 이름의 포인터 형

◎ 2DArrNameAndArrPtr.c

```
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;
}
```

```
** 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
```

배열 포인터 vs. 포인터 배열

◎ ArrPtrAndPtrArr.c

```
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;
}
```

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

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

- 포인터 배열 : 포인터 변수를 요소로 갖는 배열
- 배열 포인터 : 배열을 가리키는 포인터 변수

함수 인자로 2차원 배열 전달하기

◎ 2차원 배열을 함수 인자로 전달하기

- 배열 포인터를 사용하여 함수의 인자로 2차원 배열의 주소를 전달

```
int main(void)
{
    int arr1[2][7];   <----------> int (*parr1)[7]
    double arr2[4][5]; <----------> double (*parr2)[5]
    SimpleFunc(arr1, arr2);
    . . . .
}
```

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

동일한 선언 동일한 선언

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

함수 인자로 2차원 배열 전달하기

◎ 2DArrParam.c

```c
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;
}
```

```c
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};

                                                    배열의 세로길이 계산방식
    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;
}
```

```
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
```

2차원 배열에서 포인터를 이용한 배열 요소의 접근

◎ 배열 이름과 배열 요소의 관계

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

◎ 2차원 배열 이름을 이용한 2차원 배열 요소의 접근

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

2차원 배열에서 포인터를 이용한 배열 요소의 접근

◎ 2DArrAccessType.c

```
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;
}
```

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

6, 6
6, 6
6, 6
```