배열 포인터 처리

한국기술교육대학교
온라인평생교육원

학습내용

- 포인터 배열 기초
- 고급 포인터 배열 기술

학습목표

- 배열을 포인터 배열로 참조하는 방법을 구현할 수 있다.
- 구조체를 포인터 배열로 참조하는 방법을 구현할 수 있다.

포인터 배열 기초

1 1차원 포인터 배열

주소를 저장하는 배열

형식 데이터형* 배열명[배열 크기];

예제

```
int*        arr1 [ 5 ];   → 크기가 5인 int*형 배열
char*       arr2 [ 10 ];  → 크기가 10인 char*형 배열
double*     arr3 [ 4 ];   → 크기가 4인 double*형 배열
STUDENT*    arr4 [ 3 ];   → 크기가 3인 STUDENT*형 배열
```

```
int* arr[5];   ——— 크기가 5인 int*형 배열을 선언함
```

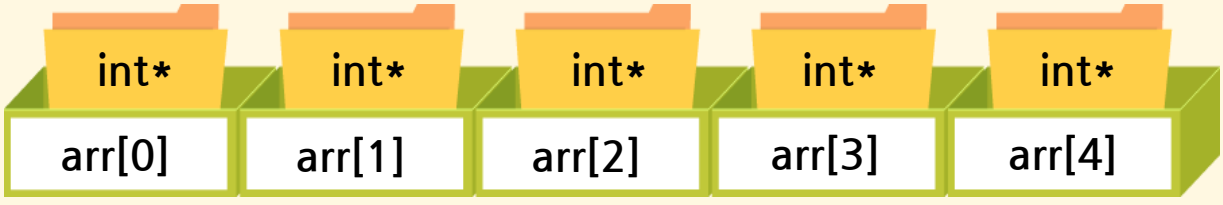
1. 포인터 배열의 각 원소로 변수의 주소를 저장함
2. 배열의 각 원소가 포인터형이므로, 원소가 가리키는 변수에 접근하려면 **배열의 원소 앞에 간접 참조 연산자 *를 사용**해야 함

포인터 배열 기초

1 1차원 포인터 배열

Int* arr[5]; ——— 크기가 5인 int*형 배열을 선언함

int* arr[5];




```
int a = 10, b = 20, c = 30, d = 40, e = 50;
int* arr[5] = {&a, &b, &c, &d, &e};
int i;
for( i = 0 ; i < 5 ; i++ )
    printf("%d", *arr[i]) ;
```

int 변수의 주소로 arr 배열의 원소를 초기화함

arr[i]는 int*형이므로 간접 참조 연산자를 사용할 수 있음

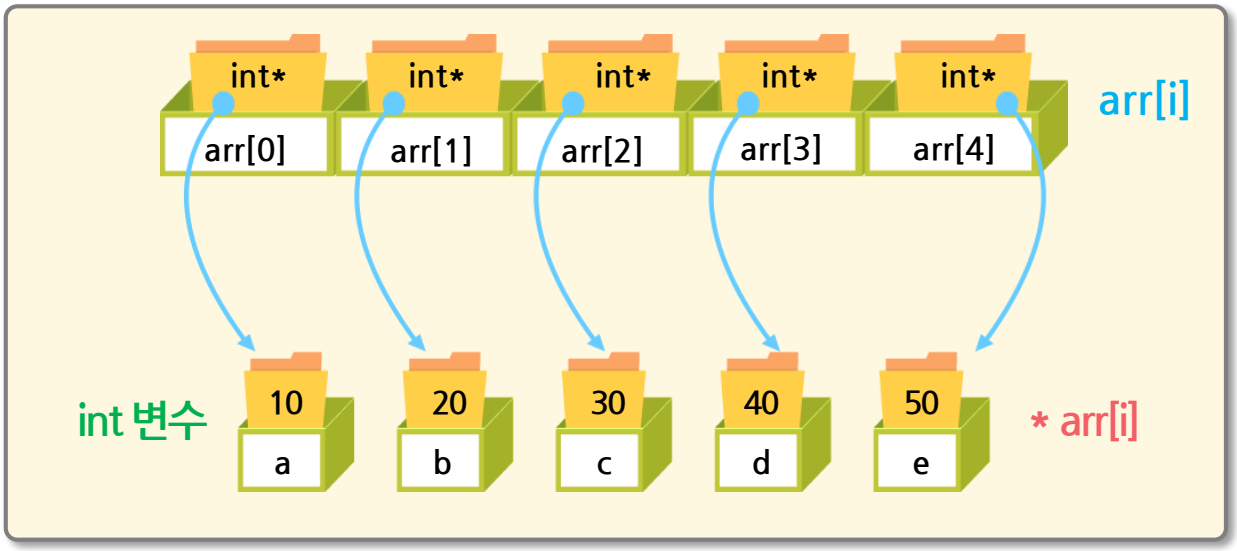
포인터 배열 기초

1 1차원 포인터 배열

```
int a=10, b=20, c=30, d=40, e=50;
int* arr[5] = {&a, &b, &c, &e, &e};
```

*arr[i]
int* 변수

int 변수




```
int main()
{
    int a=1, b=3, c=5, d=7, e=9;
    int* arr[5] = {&a, &b, &c, &d, &e};
    int i;

    for( i = 0 ; i < 5 ; i++ )
        printf("%d", *arr[i]);
    printf("₩n");

    return 0;
}
```

포인터 배열 기초

2 2차원 포인터 배열

포인터 배열의 각 원소에 배열의 시작 주소를 저장

```
int x[3] = {1, 2, 3};
int y[3] = {4, 5, 6};
int z[3] = {7, 8, 9};
int* arr[3] = {x, y, z};
```

포인터 배열의 원소를 int 배열의 시작 주소로 초기화함

arr[i]가 int 배열의 시작 주소로 초기화되었을 때,
arr[i]가 가리키는 배열의 원소에 접근하려면 arr[i][j]라고 씀

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

*(arr[i]+j)와 같은 의미

포인터 배열 기초

2 2차원 포인터 배열

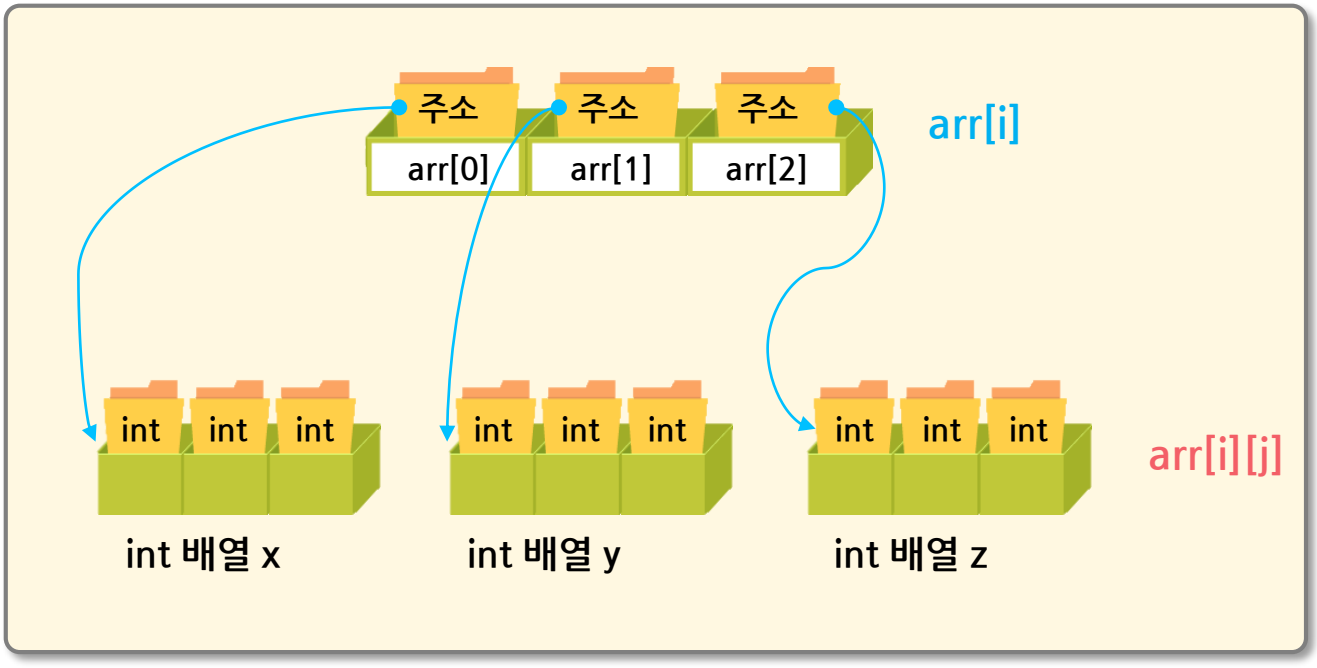
```
int x[3] = {1, 2, 3};
int y[3] = {4, 5, 6};
int z[3] = {7, 8, 9};
int* arr[3] = {x, y, z};
```

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

arr[i] : int* 변수

arr[i][j] : int 변수

*(arr[i] + j) : int 변수



포인터 배열 기초

2 2차원 포인터 배열

```
int main(void)
{
        int x[3] = {1, 2, 3};
        int y[3] = {4, 5, 6};                    int z[3] = {7, 8, 9};
        int* arr[3] = {x, y, z};
        int i, j;

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

        return 0;
}
```

고급 포인터 배열 기술

1 구조체 포인터 배열

1 구조체 배열은 메모리를 많이 사용하므로 비효율적임

```
typedef struct student {
        char name[20];
        int korean, english, math;
        double average;
} STUDENT;
```

```
STUDENT std[ 100 ];
```

STUDENT 구조체가 100개 할당되므로 40X1000바이트가 필요함

2 구조체 포인터 배열을 이용하면 구조체는 동적 메모리에 할당하고 그 주소만 포인터 배열에 넣어두고 사용할 수 있음

```
STUDENT* std[100];
```

주소만 100개 할당하므로 4x100바이트가 필요함

고급 포인터 배열 기술

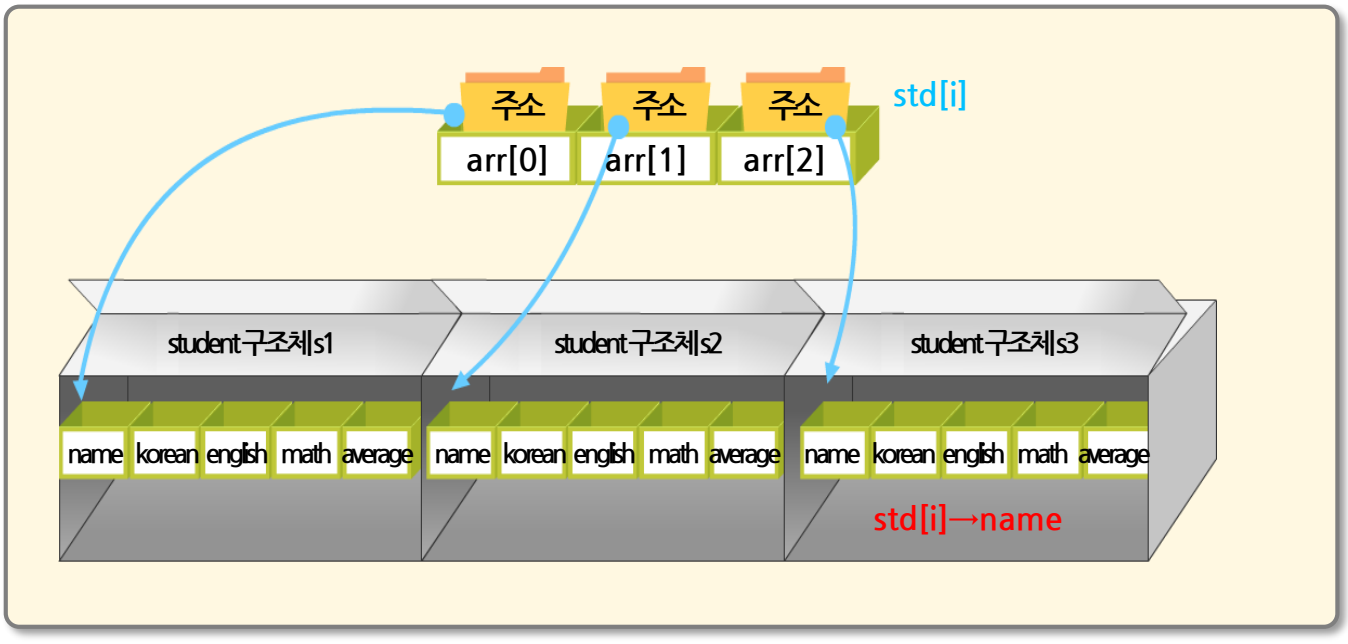
1 구조체 포인터 배열

3 구조체 포인터 배열의 메모리 구조

```
STUDENT s1;
STUDENT s2;
STUDENT s3;
STUDENT* std[3] = {&s1, &s2, &s3};

printf("%s", std[i] → name);
```

STUDENT* 변수



고급 포인터 배열 기술

2 2차원 배열 포인터 처리

행 단위 포인터 변수

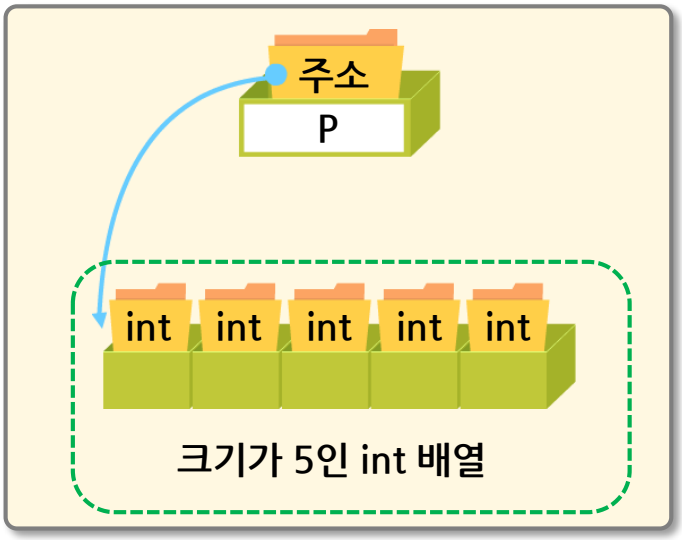
형식 데이터형 (*포인터명)[배열 크기];

예제

```
int      (*p1) [ 5 ];   → int[5] 배열을 가리키는 포인터
char     (*p2) [ 10 ];  → char[10] 배열을 가리키는 포인터
double   (*p3) [ 4 ];   → double[4] 배열을 가리키는 포인터
STUDENT  (*p4) [ 3 ];   → STUDENT[3]배열을가리키는포인터
```

int (*p)[5]; ——— 크기가 5인 int 배열을 가리키는 포인터

```
int (*p)[5];
```

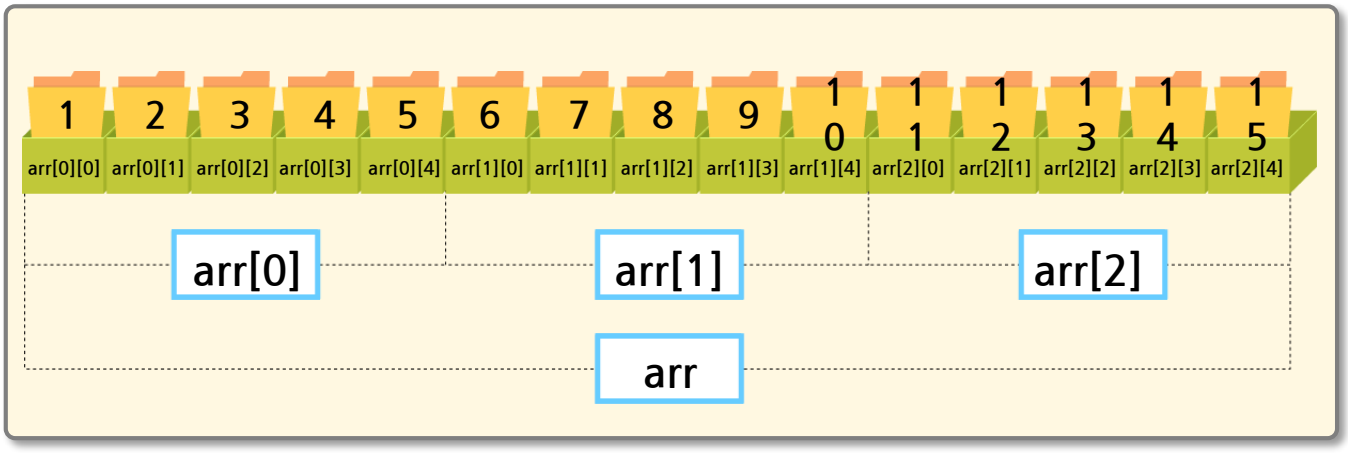


고급 포인터 배열 기술

2 2차원 배열 포인터 처리

```
int arr[3][5] = {
    { 1, 2, 3, 4, 5 },
    { 6, 7, 8, 9, 10 },
    { 11, 12, 13, 14, 15 }
};
int (*p)[5] = &arr[0];
```

p를 int 5개 짜리 배열인 arr[0]의 주소로 초기화함



배열에 대한 포인터와 이차원 배열

1 배열에 대한 포인터를 &arr[0]으로 초기화하는 대신, 간단하게 arr로 초기화할 수 있음

```
int (*p)[5] = arr;
```

arr는 &arr[0]과 같은 의미임

고급 포인터 배열 기술

2 2차원 배열 포인터 처리

배열에 대한 포인터와 이차원 배열

2 배열에 대한 포인터 p로 이차원 배열의 원소에 접근하려면 p가 마치 이차원 배열명인 것처럼 2개의 인덱스를 사용하면 됨

```
int i, j;
for ( i = 0 ; i < 3 ; i++ )
{
      for ( j = 0 ; j < 5 ; j++ )
           printf("%d", p[i][j]);
      printf(" \ n")
}
```

p[i][j]는 이차원 배열의 원소 arr[i][j]를 의미함

```
int arr[3][5];
int (*p)[5] = arr;
```

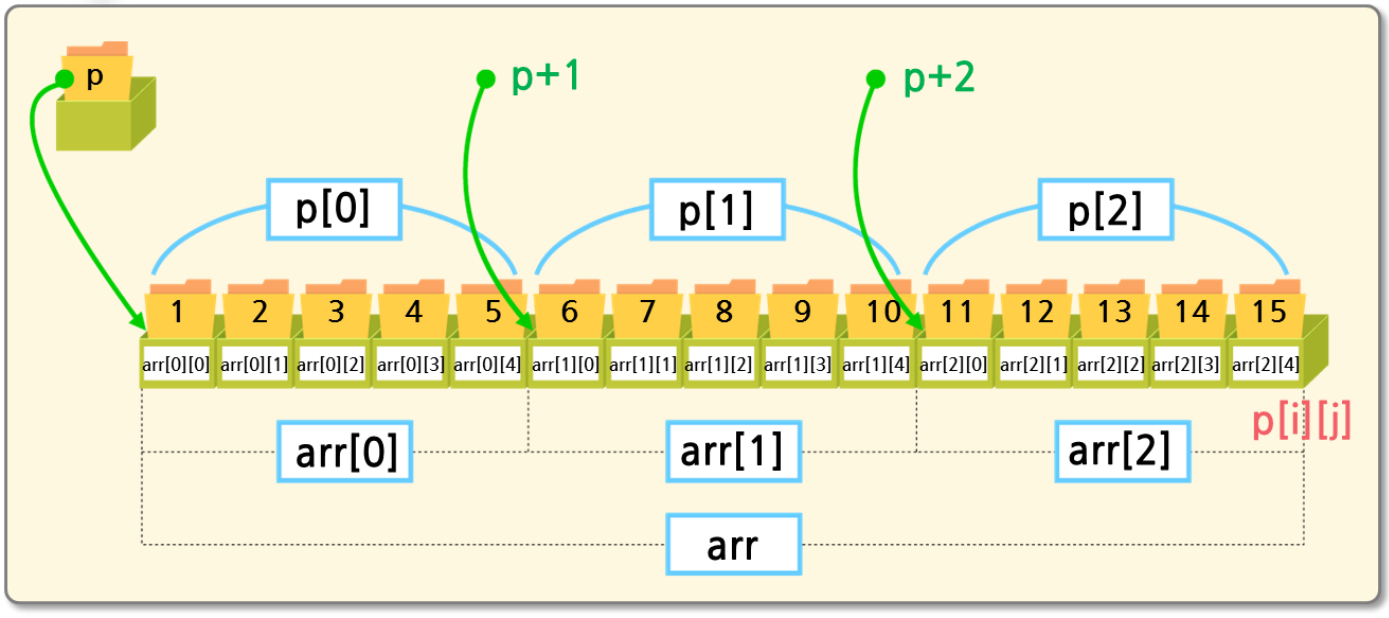
*(*(p + I) + j) = = p [i][j]

arr[i] arr[i]

arr[i][j] arr[i][j]

고급 포인터 배열 기술

2 2차원 배열 포인터 처리



학습정리

1. 포인터 배열 기초

- 각 변수의 주소를 배열과 같이 묶어서 처리하는 것이 가능함
- 배열의 주소를 포인터 배열에 저장하면 2차원 배열과 같이 참조할 수 있음

2. 고급 포인터 배열 기술

- 구조체 포인터 배열은 구조체를 이용하는 것보다 메모리 사용 측면에서 더 효율적임
- 2차원 배열은 열 크기를 기준으로 포인터 변수를 선언할 수 있음