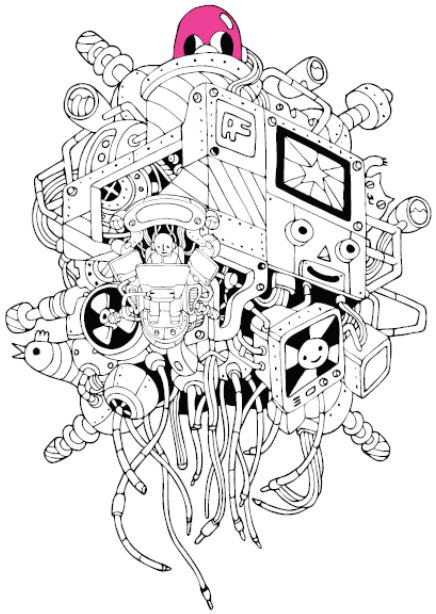
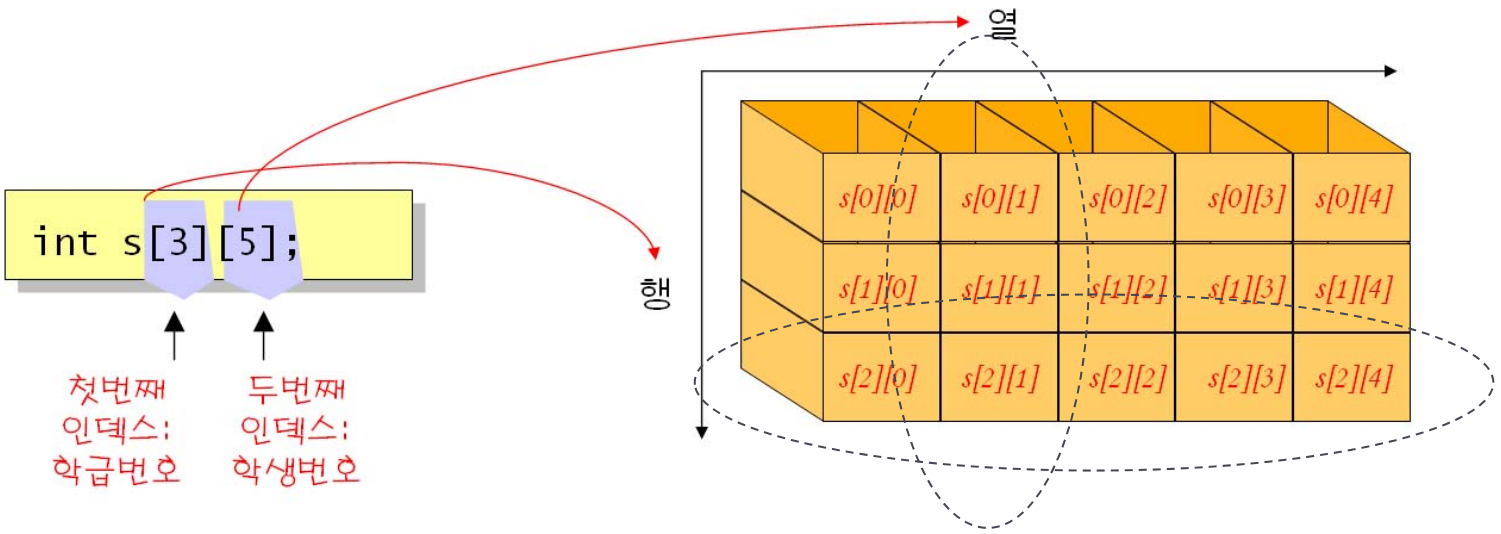
C 프로그래밍



다차원 배열

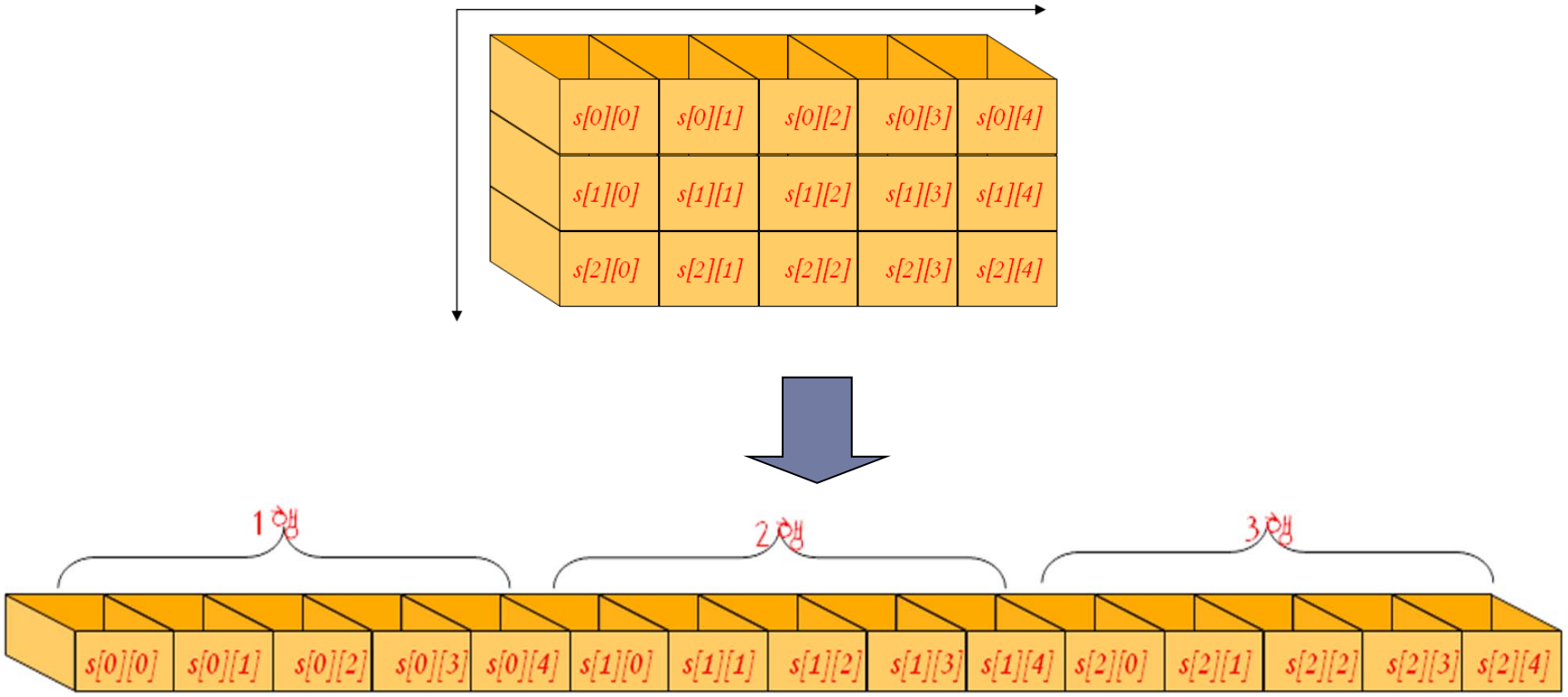
2차원 배열

```
int s[10];      // 1차원 배열
int s[3][10];   // 2차원 배열
int s[5][3][10]; // 3차원 배열
```



2차원 배열의 구현

- 2차원 배열은 1차원적으로 구현된다.



다차원 배열을 의미하는 2차원 배열의 선언

2차원 배열의 선언 방식 → TYPE arr[세로길이][가로길이];

int arr1[3][4];

int arr2[2][6];

```
int main(void)
{
    int arr1[3][4];
    int arr2[7][9];
    printf("세로3, 가로4: %d \n", sizeof(arr1));
    printf("세로7, 가로9: %d \n", sizeof(arr2));
    return 0;
}
```

실행결과

```
세로3, 가로4: 48
세로7, 가로9: 252
```

2차원 배열요소의 접근

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

	1열	2열	3열
1행	0	0	0
2행	0	0	0
3행	0	0	0

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

	1열	2열	3열
1행	1	0	0
2행	0	0	0
3행	0	0	0

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

	1열	2열	3열
1행	1	2	0
2행	0	0	0
3행	0	0	0

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;
```

	1열	2열	3열
1행	1	2	0
2행	0	0	0
3행	0	5	0

2차원 배열의 초기화

```
int s[3][5] = {
  { 0, 1, 2, 3, 4 }, // 첫 번째 행의 원소들의 초기값
  { 10, 11, 12, 13, 14 }, // 두 번째 행의 원소들의 초기값
  { 20, 21, 22, 23, 24 } // 세 번째 행의 원소들의 초기값
};
```

2차원 배열의 메모리상 할당의 형태

```
0x1001번지, 0x1002번지, 0x1003번지, 0x1004번지, 0x1005번지 . . . .
```

1차원적 메모리의 주소 값

```
0x12-0x24번지, 0x12-0x25번지, 0x12-0x26번지, 0x12-0x27번지 . . . .
0x13-0x24번지, 0x13-0x25번지, 0x13-0x26번지, 0x13-0x27번지 . . . .
0x14-0x24번지, 0x14-0x25번지, 0x14-0x26번지, 0x14-0x27번지 . . . .
. . . .
```

2차원적 메모리의 주소 값

실제 메모리는 1차원의 형태로 주소 값이 지정이 된다.

따라서 아래와 같은 형태로 2차원 배열의 주소 값이 지정된다.

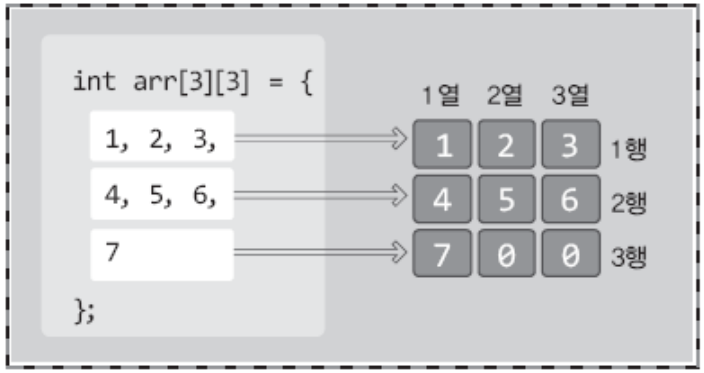
2차원 배열의 실제 메모리 할당형태

실행결과

```
002AFD54
002AFD58
002AFD5C
002AFD60
002AFD64
002AFD68
```

```c
int main(void)
{
    int arr[3][2];
    int i, j;
    for(i=0; i<3; i++)
        for(j=0; j<2; j++)
            printf("%p \n", &arr[i][j]);
    return 0;
}
```

2차원 배열 선언과 동시에 초기화 하기2



별도의 중괄호를 사용하지 않으면 좌 상단부터 시작해서 우 하단으로 순서대로 초기화된다.

한 줄에 표현해도 된다.

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

마찬가지로 빈 공간은 0으로 채워진다.

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

2차원 배열 선언과 동시에 초기화 하기(예제)

```
int main(void)
{
    int i, j;

    /* 2차원 배열 초기화의 예 1 */
    int arr1[3][3]={
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    /* 2차원 배열 초기화의 예 2 */
    int arr2[3][3]={
        {1},
        {4, 5},
        {7, 8, 9}
    };
    /* 2차원 배열 초기화의 예 3 */
    int arr3[3][3]={1, 2, 3, 4, 5, 6, 7};

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

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

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

실행결과

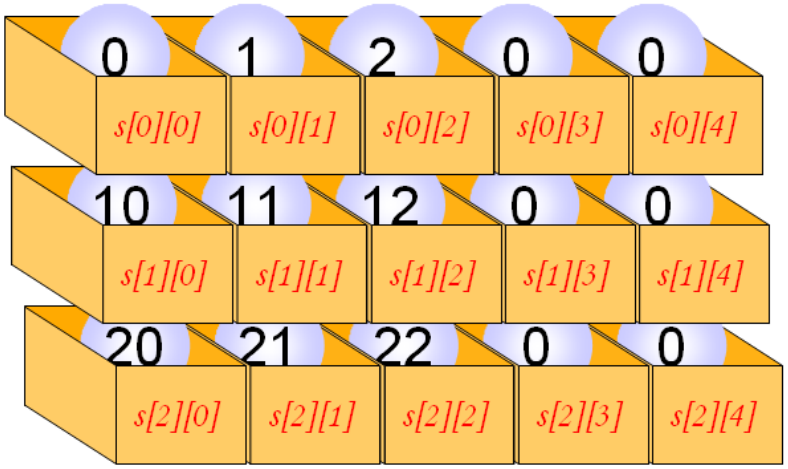
```
1 2 3
4 5 6
7 8 9

1 0 0
4 5 0
7 8 9

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

2차원 배열의 초기화

```
int s[ ][5] = {
  { 0, 1, 2, 3, 4 }, // 첫 번째 행의 원소들의 초기값
  { 10, 11, 12, 13, 14 }, // 두 번째 행의 원소들의 초기값
  { 20, 21, 22, 23, 24 }, // 세 번째 행의 원소들의 초기값
};
```

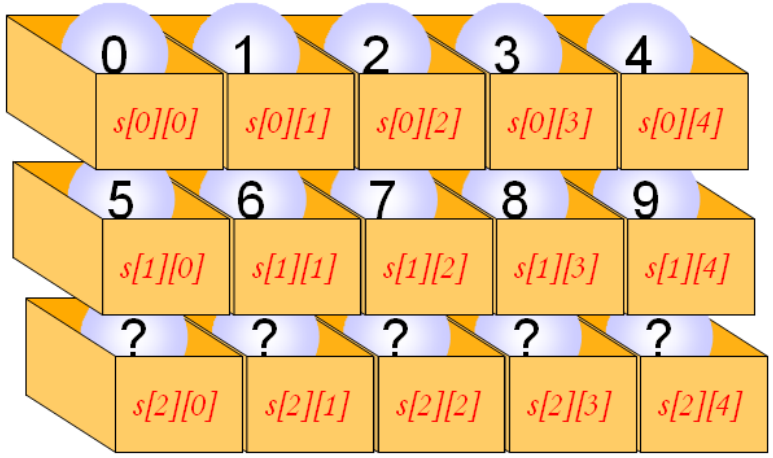
2차원 배열의 초기화

```
int s[ ][5] = {
  { 0, 1, 2 },      // 첫 번째 행의 원소들의 초기값
  { 10, 11, 12 },     // 두 번째 행의 원소들의 초기값
  { 20, 21, 22 }        // 세 번째 행의 원소들의 초기값
};
```



2차원 배열의 초기화

```
int s[ ][5] = {
  0, 1, 2, 3, 4,        // 첫 번째 행의 원소들의 초기값
  5, 6, 7, 8, 9,        // 두 번째 행의 원소들의 초기값
};
```



배열의 크기를 알려주지 않고 초기화하기

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

두 개가 모두 비면 컴파일러가 채워 넣을 숫자를 결정하지 못한다.

8 by 1 ??

4 by 2 ??

2 by 4 ??

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

세로 길이만 생략할 수 있도록 약속되어 있다.

컴파일러가 세로 길이를 계산해 준다.

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

2차원 배열요소 접근관련 예제

```
int main(void)
{
    int villa[4][2];
    int popu, i, j;

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

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

```
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
```

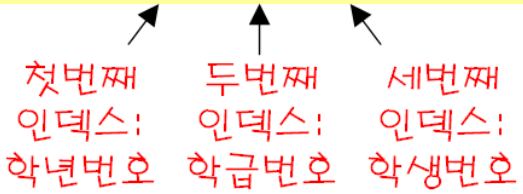
실행결과

C 프로그래밍

3차원 배열

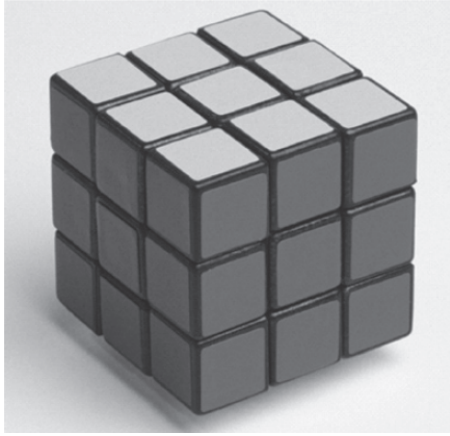
3차원 배열

```
int s [6][3][5];
```




```
#include <stdio.h>


int main(void)
{
	int s[3][3][3];	// 3차원 배열 선언
	int x, y, z;	// 3개의 인덱스 변수
	int i = 1;	// 배열 원소에 저장되는 값

	for(z=0;z<3;z++)
		for(y=0;y<3;y++)
			for(x=0;x<3;x++)
				s[z][y][x] = i++;
	return 0;
}
```

3차원 배열의 논리적 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));
    return 0;
}
```

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

실행결과

int arr1[2][3][4];

높이 2, 세로 3, 가로 4인 int형 3차원 배열(세로 3, 가로 4인 배열이 두 개 겹친 형태)

double arr2[5][5][5];

높이, 세로, 가로가 모두 5인 double형 3차원 배열(세로 5, 가로 5인 배열이 5개 겹친 형태)

3차원 배열의 선언과 접근

```
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);
    return 0;
}
```

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

실행결과

윤성우의

열혈 C 프로그래밍

Chapter 18-1. 2차원 배열이름의 포인터 형

윤성우 저 열혈강의 C 프로그래밍 개정판

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 **)

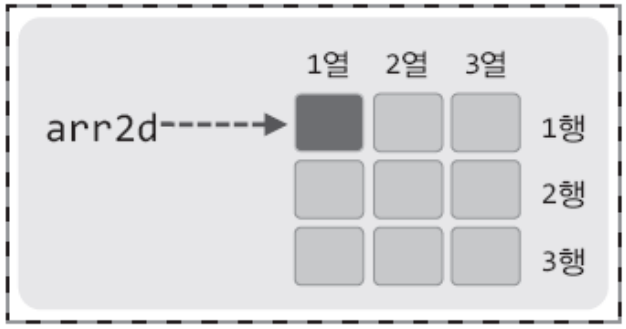
```
int arr2d[3][4];
```

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.

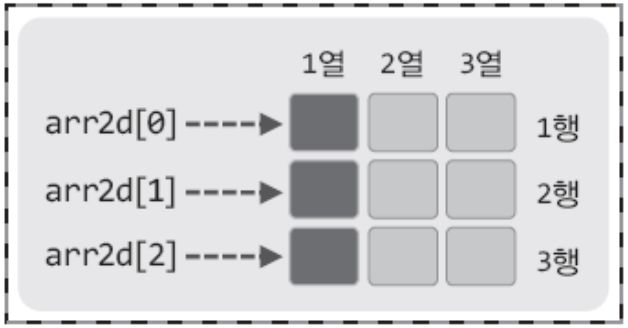
2차원 배열이름이 가리키는 것들은?

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 주소값의 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);
```

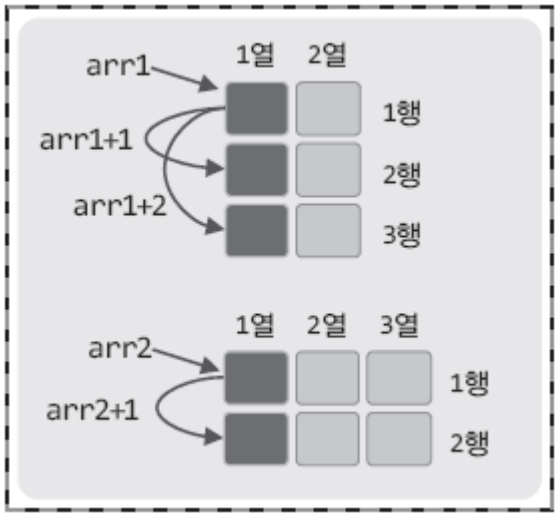
iarr은 int형 포인터이기 때문에 +1의 결과로 sizeof(int)의 크기만큼 값이 증가한다.

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.

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: 004BFBCC
```

실행결과

arr1과 arr2는 둘 다 int형 2차원 배열이다. 그러나 가로의 길이가 다르기 때문에 포인터 연산결과로 증가 및 감소하는 값의 크기에는 차이가 있다. 즉, arr1과 arr2는 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.

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?

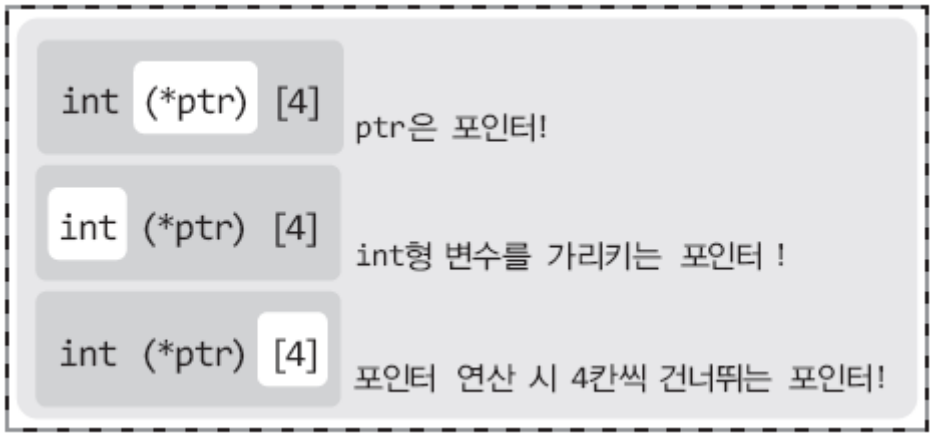
어색하지만 이것이 arr의 포인터 형을 설명하는 최선의 방법이다.
int형 포인터, double형 포인터와 같이 딱 떨어지는 명칭이 존재하지 않는다.

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 변수
이므로 배열 포인터 변수라 한다.



2차원 배열이름의 포인터 형 결정짓는 연습

case 1

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

"arr1은 char형 변수를 가리키면서, 포인터 연산 시 sizeof(char)×4의 크기단위로 값이 증가 및 감소하는 포인터 변수"

"arr2는 double형 변수를 가리키면서, 포인터 연산 시 sizeof(double)×7의 크기단위로 값이 증가 및 감소하는 포인터 변수"

case 2

"int형 변수를 가리키면서, 포인터 연산 시 sizeof(int)×2의 크기단위로 값이 증가 및 감소하는 포인터 변수 ptr1"

"float형 변수를 가리키면서, 포인터 연산 시 sizeof(float)×5의 크기단위로 값이 증가 및 감소하는 포인터 변수 ptr2"

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

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
```

윤성우의

열혈 C 프로그래밍

Chapter 18-2. 2차원 배열이름의 특성과 주의사항

윤성우 저 열혈강의 C 프로그래밍 개정판

‘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);
    . . . .
}
```

int (*parr1)[7]

double (*parr2)[5]

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] ) { . . . . }
```

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};

                                          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;
}
```

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!

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

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

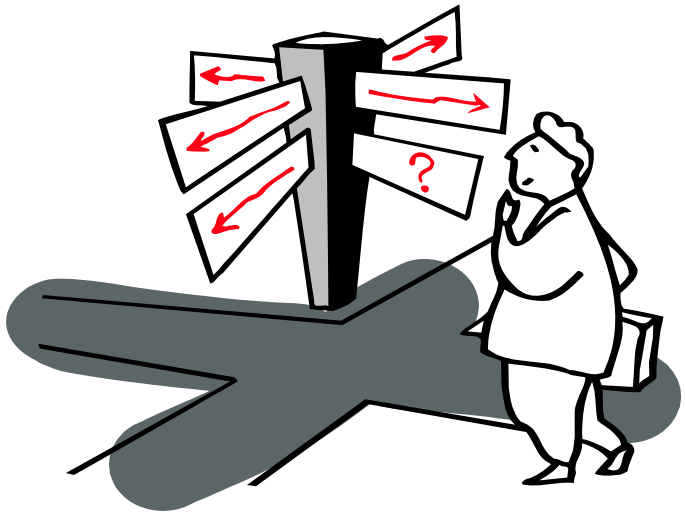
실행결과

```
int 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.

```
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;
}
```



Chapter 16이 끝났습니다. 질문 있으신지요?