10주차 1차시 다차원배열

【학습목표】

1. const 선언에 대해 설명할 수 있다.
2. 다차원배열에 대해 설명할 수 있다.

학습내용1 : 포인터 대상의 const 선언

1. 포인터 변수의 참조 대상에 대한 const 선언

```
int main(void)
{
    int num=20;
    const int * ptr=&num;
    *ptr=30;    // 컴파일 에러!
    num=40;     // 컴파일 성공!
    . . . .
}
```

위의 *const* 선언이 갖는 의미
포인터 변수 ptr을 이용해서 ptr이 가리키는 변수에 저장된 값을 변경하는 것을 허용하지 않겠습니다!
그러나 변수 num에 저장된 값 자체의 변경이 불가능한 것은 아니다.
다만 ptr을 통한 변경을 허용하지 않을뿐이다.

2. 포인터 변수의 상수화

```
int main(void)
{
    int num1=20;
    int num2=30;
    int * const ptr=&num1;
    ptr=&num2;      // 컴파일 에러!
    *ptr=40;        // 컴파일 성공!
    . . . .
}
```

위의 *const* 선언이 갖는 의미
포인터 변수 ptr에 저장된 값을 상수화 하겠다. 즉, ptr에 저장된 값은 변경이 불가능하다.
ptr이 가리키는 대상의 변경을 허용하지 않는다.

```
const int * ptr=&num;
int * const ptr=&num;
```

→

```
const int * const ptr=&num;
```

두 가지 const 선언을 동시에 할 수 있다.

3. const 선언이 갖는 의미

√const 선언은 추가적인 기능을 제공하기 위한 것이 아니라, 코드의 안전성을 높이기 위한 것이다. 따라서 이러한 const의 선언을 소홀히 하기 쉬운데, const의 선언과 같이 코드의 안전성을 높이는 선언은 가치가 매우 높은 선언이다.

```
int main(void)
{
    double PI=3.1415;
    double rad;
    PI=3.07;   // 실수로 잘못 삽입된 문장, 컴파일 시 발견 안됨
    scanf("%lf", &rad);
    printf("circle area %f \n", rad*rad*PI);
    return 0;
}
```

```
int main(void)
{
    const double PI=3.1415;
    double rad;
    PI=3.07;   // 컴파일 시 발견되는 오류상황
    scanf("%lf", &rad);
    printf("circle area %f \n", rad*rad*PI);
    return 0;
}
```

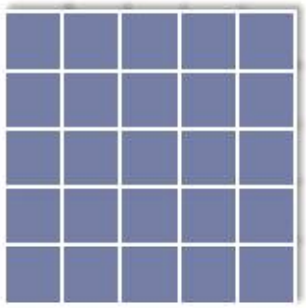
학습내용2 : 다차원배열의 이해와 활용

1. 2차원, 3차원 배열 OK, 4차원, 5차원 배열 NO!

```
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

문법적으로는 4차원 5차원 배열의 선언도 가능하지만 그것은 의미를 부여하기 힘든, 의미가 없는 배열이다.

2. 다차원배열을 의미하는 2차원 배열의 선언

2차원 배열의 선언방식 → 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
```

3. 2차원 배열요소의 접근

√일반화

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

세로 N, 가로 M의 위치에 값을 저장 및 참조

4. 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
```

5. 2차원 배열의 메모리상 할당의 형태

```
0x1001번지, 0x1002번지, 0x1003번지, 0x1004번지, 0x1005번지 . . . .
                1차원적 메모리의 주소값

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 주소값
```

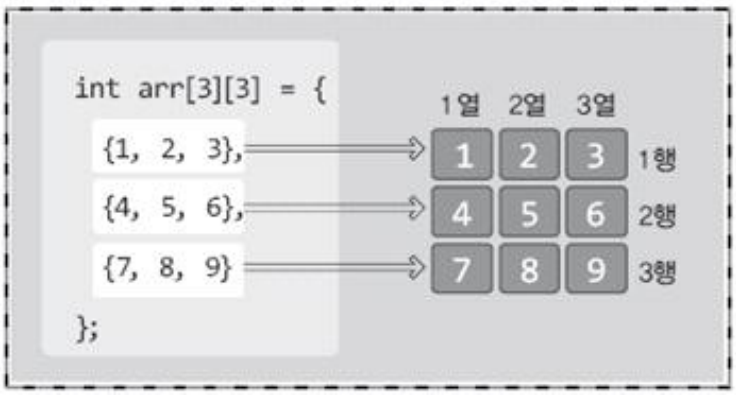
실제 메모리는 1차원의 형태로 주소 값이 지정이 된다.
따라서 아래와 같은 형태로 2차원 배열의 주소 값이 지정된다.

2차원 배열의 실제 메모리 할당형태

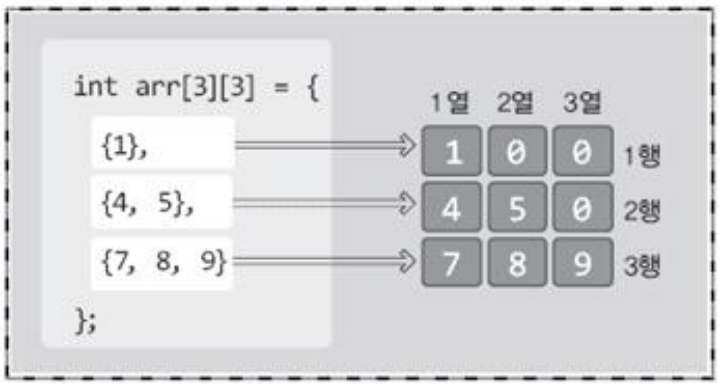
```
int main(void)
{
    int arr[3][2];
    int i, j;
    for(i=0; i<3; i++)
        for(j=0; j<2; j++)
            printf("%p \n", &arr[i][j]);
    return 0;
}
```

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

6. 2차원 배열 선언과 동시에 초기화 하기



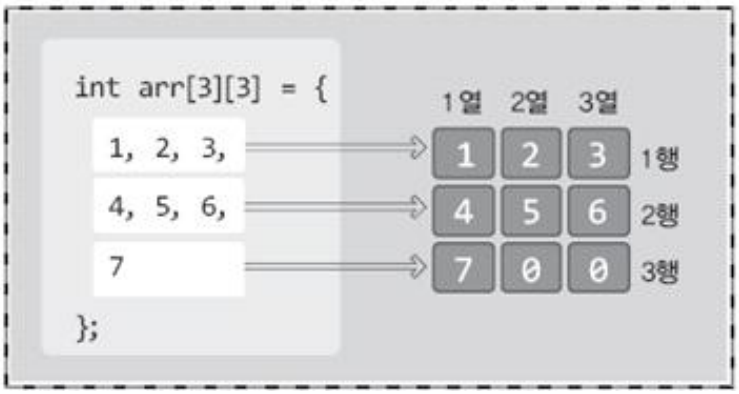
초기화 리스트 안에는 행 단위로 초기화할 값들을 별도의 중괄호로 명시한다.




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

채워지지 않은 빈 공간은 0으로 채워진다.

7. 2차원 배열 선언과 동시에 초기화 하기2



별도의 중괄호를 사용하지 않으면 좌 상단부터 시작해서 우 하단으로 순서대로 초기화된다.
한 줄로 표현하면 int arr[3][3]={1,2,3,4,5,6,7} 마찬가지로 빈공간은 0으로 채워진다.
int arr[3][3]={1,2,3,4,5,6,7,0,0}

8. 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
```

9. 배열 크기를 알려주지 않고 초기화 하기

```
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};
```

【학습정리】

1. 2차원 배열의 메모리상 할당의 형태는 실제 메모리는 1차원의 형태로 주소 값이 지정이 된다.

2. 배열의 크기를 알려주지 않고 초기화 하면 컴파일러가 숫자를 결정하지 못한다.

3. 2차원 배열은 세로의 길이만 생략할 수 있도록 약속되어 있다. 빈 칸은 컴파일러가 채운다.

4. 2차원 배열은 별도의 중괄호를 사용하지 않으면 좌 상단부터 시작해서 우 하단으로 순서대로 초기화된다.