# C 언어 및 Java 언어의 배열 개념의 비교 분석

류기열 (아주대학교 소프트웨어학과 교수, kryu@ajou.ac.kr)

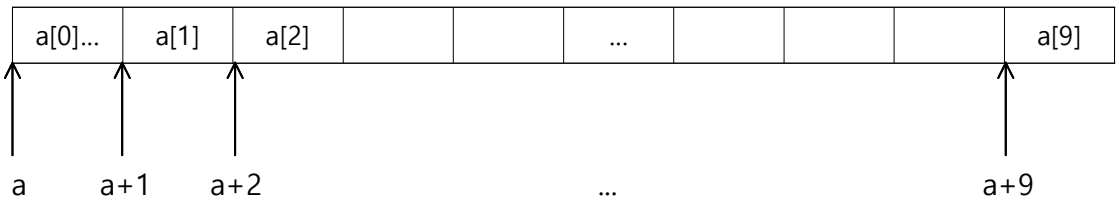
## 1. 들어가며

C 언어와 Java 언어의 배열이 개념적으로 유사성이 많으면서 차이점도 많다. 우선 Java 배열의 문법적 체계는 C 언어의 그것을 모방하였다. a[i][j]와 같이 배열을 접근하는 연산자([])가 대표적이다. 또한 다차원 배열의 경우 **배열의 배열** 개념 또한 닮아있다. 가령, 2차원 배열은 1차원 배열의 1차원 배열이다. 즉, 2차원 배열은 1차원 배열을 원소로 가지는 1차원 배열이다. 3차원 배열은 2차원 배열을 원소로 가지는 1차원 배열이다. 즉 모든 배열은 1차원 배열이라고 해도 무방하다. 이 부분은 뒤에서 보다 자세하게 다룰 예정이다. 하지만 두 언어의 배열은 **타입의 관점이나 구현의 관점에서 크게 다르다**. C 언어의 배열은 메모리 주소 개념과 매우 깊은 관계가 있으며 주소를 이용하여 다룰 수 있는 반면, Java 언어의 배열은 객체로 만들어져 데이터가 캡슐화되어 있어 데이터를 처리하기 위해서는 주어진 연산자(가령, [])와 메소드를 이용하지 않고서는 다룰 수 없다.

## 2. C 언어의 배열

우선 C 언어의 배열 타입과 구현 개념을 이해하기 위해서는 **배열과 포인터와의 상관관계를 잘 이해해야 한다.** 앞으로 사용하는 예시는 모두 1차원 배열은 int a[10]을, 2차원 배열은 int b[3][4]를 이용하기로 한다.

배열의 이름 a는 그 시작 원소(a[0])의 주소값(상수)이다. a+1은 두 번째 원소, a+2는 세 번째 원소를 가리킨다. <그림 1>은 배열과 주소와의 상관관계를 보여준다.




<그림 1> 배열과 주소와의 관계

배열 이름이 주소이니까 배열 원소는 배열 이름을 이용한 dereference 연산자(*)를 이용하여 접근이 가능하다.

a[0] = *(a+0)

a[1] = *(a+1)

a[2] = *(a+2)

...

**배열은 단지 시작 주소에 불과**하다. 즉, 원소에 대한 접근, 저장, 매개변수 전송 등 모든 배열에 대한 작업은 모두 시작 주소를 이용한 연산으로 가능하다.

**그렇다면 배열의 타입은 무엇인가?** 좀 더 정확하게는 배열 이름의 타입이 무엇인가라고 하는 게 올바른 표현이다. 배열 이름이 첫 번째 원소(여기서는 int 형)의 주소를 나타내니 그 타입은 int * 형 (int 형 변수의 주소)이다. 포인터와 배열과의 관계를 좀 더 이해하기 위해 다음과 같이 정의해보자.

int *p = a;

p의 타입이 int *이고 a의 타입도 int *이니 assignment가 가능하다. 이제 p는 a와 동일한 값(즉, a[0]의 주소)을 가지고 있다. 따라서 p도 하나의 배열로 간주할 수 있다. p 배열의 첫 번째 원소는 a[5]와 동일하다. 따라서 다음이 성립한다.

a[0] = *(a+0) = *(p+0) = p[0]

a[1] = *(a+1) = *(p+1) = p[1]

...

좀 더 나가보자.

p = a+5;

즉, p가 a[5]를 가리키니 다음이 성립한다.

a[5] = *(a+5) = *(p+0) = p[0]

a[6] = *(a+6) = *(p+1) = p[1]

...

즉 p 배열은 a[5]부터 시작하는 배열이다. 결론적으로 **배열은 어떤 주소값을 시작으로 하는 연속적인 메모리 공간**인 것이다.

다음으로 **배열의 공간적인 측면**을 살펴보자. 배열의 크기는 무엇인가? int a[10]은 10개의 정수 크기의 공간(4바이트*10=40바이트)을 연속적으로 확보한다. 하지만 이 선언문은 컴파일러에게 메모리 공간을 확보하라는 지시어로 실행과는 무관하다. 즉, **실행시 배열의 크기 정보는 어디에도 없다**. 단지 프로그램에서 10개의 정수 공간을 마련했으니 접근할 때도 a[0]에서 a[9]까지 사용하는 것이 안전할 뿐이다. 앞의 예에서 a[10]이라든지 p[5]는 a를 선언하면서 확보한 범위를 벗어나 있지만 문법적으로는 전혀 문제가 없다. 다만, (a+10이나 (p+5)가 가리키는 영역이 사용자의 접근

이 허용되지 않는 공간(가령 코드 영역이라든지 시스템 영역 등)이라면 실행시 runtime 오류가 발생한다. 또한 그 공간이 허용된 공간이라면 엉뚱하게 데이터를 파괴할 수도 있다. 실제로 전문가들은 의도적으로 이러한 기법을 써서 프로그래밍을 하기도 한다.

### 3. Java 언어의 배열

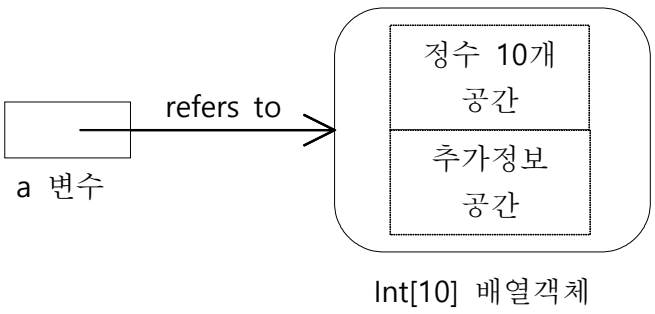
이제 Java 언어의 배열을 C 언어와 비교하면서 살펴보자. int형 배열을 사용하기 위해서는 우선 배열을 위한 변수를 다음과 같이 선언한다.

```
int[] a;
```

여기서 우리는 흔히 a를 배열이라고 말하지만 사실 a는 배열이 아니다. 배열 객체를 저장할 수 있는 **변수**(배열변수라고 부르기로 하자)이다. **a의 타입은 int[]**로 reference 타입이다. reference 타입의 변수는 그 안에 실제 값을 저장하는 것이 아니라 객체(배열도 하나의 객체)를 가리키는 reference 값을 저장한다. a는 어떤 길이의 int형 1차원 배열 객체라도 참조할(refer to) 수 있는 변수이다. 다음과 같이 배열 객체를 생성하여 그 객체의 참조(reference) 값을 a 변수에 지정할 수 있다. new int[10]은 10개 int형을 저장할 수 있는 배열 객체를 생성한다.

```
a = new int[10];
```

공간의 크기와 모양은 어떤가? 4바이트*10=40바이트를 확보할까? C 언어가 연속적인 10개의 정수를 위한 공간을 만드는 것과는 다르게 Java에서는 하나의 객체를 위한 공간을 확보한다. 프로그래머는 객체 속의 모습은 알 수가 없고 또 알 필요도 없다. 구현한 사람만이 알고 있다. 사용자는 단지 주어진 연산자나 메소드를 통해 접근할 뿐이다. **바로 이게 객체지향언어의 캡슐화**이다. 앞에서 보았듯이 C 언어에서는 []연산자뿐만 아니라 포인터 변수를 통해서도 얼마든지 배열의 데이터를 접근하고 수정할 수도 있다. 데이터에 대한 캡슐화가 이루어지고 있지 않다는 것을 의미한다.




<그림 2> 배열 변수와 배열 객체 개념도

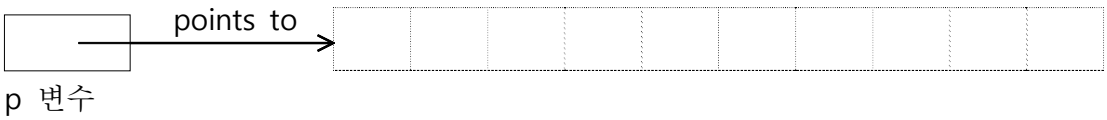
유추하건데, 배열 객체 속에는 분명 정수 10개를 저장할 공간이 있을 것이다. 이 외에도 배열의 길이, 타입 정보 등 여러 가지 추가적인 내용이 담겨있다. 일종의 구조체를 생각하면 된다. 하지만 그 구조를 알 필요는 없다. <그림 2>는 배열 변수와 배열 객체의 관계에 대한 개념도이다.

앞에서 Java 언어에서 배열은 객체로 되어 있고 a와 같은 변수가 이를 가리키고(refer to) 있다. 이 **reference는 C 언어의 주소와는 다른 개념**이다. 내부 구현의 측면에서는 유사할 수 있겠지만 가리키는 대상도 다르고 타입도 다르며 프로그램 영역에서 사용할 수 있는 연산도 다르다. 주소는 다양한 주소 연산(&, *, +, -, >, ==, etc.)이 가능하나 reference는 .(field 선택이나 메소드 적용 연산)과 같은 멤버 접근 연산만이 가능하다. 물론 []은 두 언어에 공통적으로 적용되는 배열 원소에 대한 접근 연산자이다.

또 한 가지 차이는 C 언어의 배열은 주소 상수값인 이름으로 처리하는 데 반해 Java 배열은 배열 객체를 가리키는 변수를 통해 처리한다. 즉. Java의 배열은 동적으로 생성되는 객체로 **참조 값만 있을 뿐 이름을 가지고 있지 않다**. C 언어의 배열 이름은 그 값을 변경할 수 없지만 Java 배열 객체를 가리키는 변수는 언제든지 다른 배열을 가리키도록 변경할 수 있다. 이러한 관점에서 Java의 배열 변수는 배열을 가리킬 수 있는 C 언어의 포인터 변수와 개념적으로 더 유사하다. 아래 선언에서 p 변수가 배열과 동일한 개념이라는 것은 2절에서 설명하였다.

```
int *p = (int *)malloc(40);
```

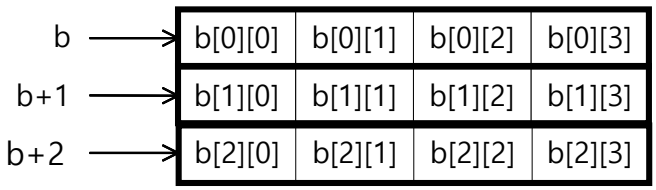
malloc 함수는 동적으로 이름 없는 배열 공간을 할당한다는 측면에서 Java에서 객체를 동적으로 생성하는 new와 유사하다.




<그림 3> C 언어 배열의 동적 할당

### 3. C 언어와 Java 언어의 다차언 배열 비교

이제 두 언어의 다차원 배열을 비교해보자. 이 절에서는 논의를 간단하게 하기 위해 2차원 배열의 경우를 예제로 사용할 것이나 3차원 이상도 비슷한 방식으로 확장해서 해석할 수 있다. Java 언어와 C 언어 모두 **2차원 배열은 1차원 배열의 1차원 배열**이다. 하지만 1차원 배열에서 보았듯이 타입과 구현의 관점에서 전혀 다르다. 먼저 C 언어에서의 2차원 배열의 개념을 살펴보자.




<그림 4> C 언어의 2차원 배열 개념도

<그림 4>는 C 언어에서 int b[3][4]를 나타낸 개념도이다. 2차원 배열 b는 각 행(1차원 배열)을 원소로 가지는 1차원 배열로 해석할 수 있다. b[0]는 첫 번째 행을 나타내는 1차원 int형 배열이고 b[1]은 두 번째 행을 위한 1차원 int형 배열이다. 그림에서 굵은 박스로 된 것이 각각 하나의 원소가 된다. 그렇다면 b는 무엇인가? 앞에서 1차원 배열에서 설명한 바와 같이 배열의 이름은 첫 번째 원소의 주소이다. 즉, b는 첫 번째 원소를 가리키는 주소인데 첫 번째 원소가 다름 아닌 첫 번째 행이 된다. 즉, b가 가리키는 대상은 하나의 정수가 아니라 하나의 행이 된다. b+1은 두 번째 원소를 가리킨다. 즉, 두 번째 행을 가리키게 된다.

그리고 b[0], b[1], b[2]는 각 행을 나타내는 정수형 배열 이름에 해당한다. int a[10]에서 a는 메모리 공간이 없이 상수인 것과 마찬가지로 b[0], b[1], b[2]도 주소 상수이다. 그러면 b는 어떤 값을 가질까? 첫 번째 원소인 b[0]의 주소 값(b[0]가 배열이므로 배열의 주소가 되는데 이는 개념적으로 모호함)으로 실제로는 b[0][0]의 주소 값과 동일하나 그 타입이 다르다. 이 부분은 아래 타입을 설명할 때 하기로 한다. 1차원 배열에서 적용했던 주소를 이용한 배열 원소의 접근을 2차원 배열에도 적용해보자. 다음과 같은 식이 성립할 것이다.

*(b+0) = b[0]

*(b+1) = b[1]

*(b+2) = b[2]

*(*(b+0)+0) = *(b[0]+0) = b[0][0]  *(*(b+0)+1) = *(b[0]+1) = b[0][1] ...

*(*(b+1)+0) =*(b[1]+0) = b[1][0]  *(*(b+1)+1) = *(b[1]+1) = b[1][1] ...

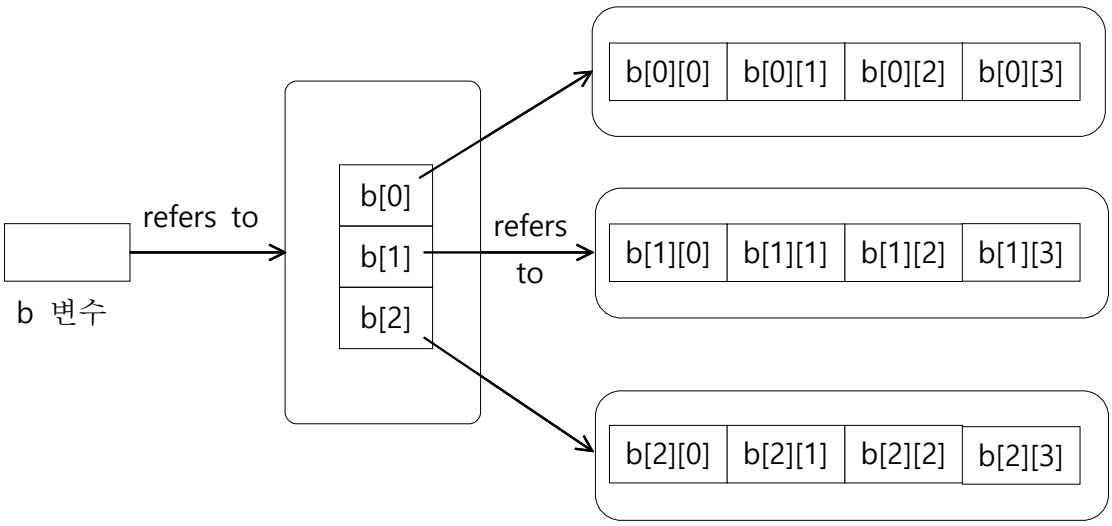
...

3차원, 4차원도 동일한 방식으로 확장해나가면 된다. 3차원 배열은 앞에서 정의한 2차원 배열들을 원소로 가지는 1차원 배열이고 4차원 배열원 3차원 배열들을 원소로 가지는 1차원 배열이다. 즉, 모든 다차원 배열은 1차원 배열이라고 간주해도 무방하다. 다만 표현하고자 하는 데이터는 1차원은 선형, 2차원은 행렬, 3차원은 공간적인 구조 등으로 차원에 따라 그 구조가 모두 다르겠지만 말이다.

다음으로 Java에서의 2차원 배열의 개념을 보자. <그림 5>는 아래 선언에 대한

배열 변수와 2차원 배열 객체의 구조를 보여주고 있다.

int[][] b = new int[3][4];




<그림 5> Java 언어에서의 2차원 배열의 개념도

Java의 2차원 배열 역시 1차원 배열을 원소로 가지는 1차원 배열의 구조로 되어 있다. C 언어와의 차이는 이러한 배열들이 객체들로 이루어져 있다는 점이다. 또한 b[0], b[1], b[2]가 각각 int형 1차원 배열에 대한 참조 값을 가지는 배열 변수라는 점에서 차이가 난다. 그리고 이들이 변수이기 때문에 실제로 b[0]에 다른 배열을 가리키도록 변경할 수도 있다. 따라서 이 초기에는 [3][4] 배열이지만 얼마든지 각 행의 길이가 다른 배열을 가지는 것도 가능하다. C에서는 b[0]와 같은 값이 주소 상수이기 때문에 [3][4]배열을 정의한 후 행을 교체할 수 없다. 하지만 뒤에서 보겠지만 C 언어에서도 행을 교체할 수 있는 배열을 정의할 수 있다.

다음으로 2차원 배열의 공간에 대해서 살펴보자. C 언어에서 2차원 배열 int b[3][4]는 12개의 연속적인 정수 공간(12*4바이트 =48바이트)을 확보한다. <그림 4>에서 직사각형으로 표시한 것은 2차원적인 구조로 이해를 돕기 위한 개념도로써 실제로는 각 행들이 연속적인 메모리 공간에 할당된다. 이렇게 연속적으로 할당되면 주소연산에 의해 쉽게 i-번째 원소를 계산하기 쉽다는 장점이 있다. Java에서는 각 행이 배열 객체로 되어 있고, 이들은 b[0], b[1], b[2]와 같은 변수에 의해 참조되고 있어 각 객체가 연속적으로 배치되지 않으며 그럴 필요도 없다. 즉, C 언어와는 다르게 주소 연산에 의해 행들의 위치가 계산되지 않고 참조를 통해 접근이 된다.

이제부터 타입의 관점에서는 어떻게 다른지 살펴보기로 하자. C 언어에서 1차원 배열 int[10]은 type이 int*이다. 2차원 배열 int[3][4]의 타입은 무엇일까? int (*p)[4]에서 변수 p와 동일한 타입이다. 즉, int (*)[4]가 int[3][4]의 타입이다. 이 타입의 의미는 원소가 4개인 1차원 배열에 대한 주소 타입이다. 즉, 1차원 배열을 가리키는

주소 값이라는 이야기다. <그림 4>에서 본 굵은 선의 박스에 대한 주소를 생각하면 된다. 여기서 주목할 것은 2차원 배열의 타입에는 행의 개수는 들어 있지 않고 열의 개수만 들어 있다. 그 이유는 배열은 첫 번째 원소에 대한 시작 주소값으로 결정되기 때문이다. 열의 개수는 원소의 타입이 '4개의 정수를 가지는' 배열이기 때문에 반드시 포함되어야 한다. 가령, 두 번째 원소(행)의 주소의 계산식은 b+4*4bytes가 된다. 바로 앞의 4가 열의 개수(행의 길이) 값이다.

각 원소인 b[0], b[1], b[2]의 타입은 각각이 int형 배열을 의미하므로 int* 타입이 된다. 앞에서 b의 실제값은 b[0][0]의 주소값과 동일하고 타입이 다르다고 했는데 b의 타입은 int(*)[4]이고 b[0][0]의 주소의 타입은 int*가 된다.

다음과 같이 p에 b를 지정하면 p가 2차원 배열이 된다. 즉, p[0][0]와 같이 접근할 수 있다.

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

p = b;
```

이를 매개변수에 적용하면 2차원 배열을 매개변수로 전달하고자 할 때 p와 같이 매개변수를 선언하면 된다.

자바의 int형 2차원 배열의 타입은 그냥 int[][]이다. C 언어와 또 다른 차이는 C 언어의 int b[3]][4] 선언에서 b는 주소상수이고 Java 언어의 int[][] b 선언에서 b는 변수이다. b가 변수이기 때문에 int형 2차원 배열이면 무엇이든 참조할 수 있다. 배열의 원소 b[0], b[1], b[2]의 타입은 int[] 형이 된다 (C 언어에서는 int *).

그럼 C 언어에서는 Java 언어에서처럼 실행 중에 특정 행을 변경할 수 있는 배열을 만들 수 없을까? 가능하다. 포인터 변수의 배열을 만들면 된다. 즉, 배열의 각 원소가 포인터 변수라면 그것이 배열을 의미하게 되어 2차원 배열이 되고, 또 포인터 변수는 언제든지 다른 배열을 지정할 수 있기 때문에 2차원 배열에서 행을 교체할 수 있게 된다.

int **b; 라고 하면 b는 int*형에 대한 주소값(int **)을 저장할 수 있다. 즉, (int *)형 배열의 시작 주소가 될 수 있다는 뜻이다. 이 배열의 각 원소가 int*이니까 이 원소는 다시 int형 1차원 배열이 될 수 있다. 즉 b는 2차원 배열이 된다. 그런데 b 배열의 각 원소가 포인터 변수이기 때문에 임의의 크기의 배열을 가리킬 수 있으니까 행의 길이가 서로 다른 배열을 만들 수 있게 된다. 이 개념은 <그림 5>에서와 같은 Java 언어의 2차원 배열을 흉내낼 수 있다.

b가 2차원 배열이 되기 위해서는 먼저 배열의 공간을 동적으로 할당해야 한다. 이를 위해서는 앞에서 본 malloc을 사용하면 된다.

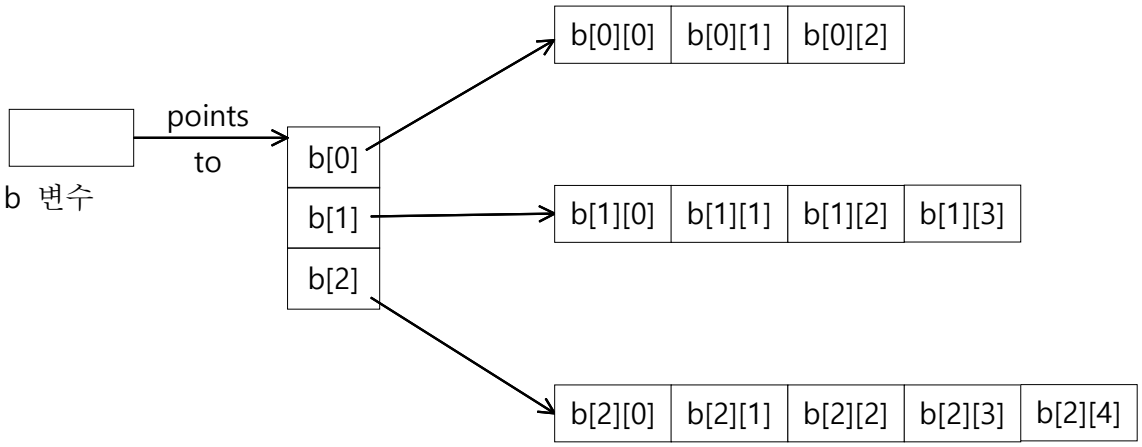
```
b = (int **)malloc(3*sizeof(int *)); // 3개의 (int *)을 가질 수 있는 공간 할당
```

b[0] = (int *)malloc(3*sizeof(int)); // 첫 번째 행(int 3개)을 위한 공간 할당

b[1] = (int *)malloc(4*sizeof(int)); // 두 번째 행(int 4개)을 위한 공간 할당

b[2] = (int *)malloc(5*sizeof(int)); // 세 번째 행(int 5개)을 위한 공간 할당

이렇게 선언한 배열의 모습은 <그림 6>과 같다.




<그림 6> C 언어에서 가변길이 행을 가진 2차원 배열 (Rugged Array)