프로그래밍 기초 with C#

04. 배열

학습목표

- 배열 객체를 다룰 수 있다.

들어가며

프로그래밍을 하다 보면 비슷한 성질을 가지는 데이터를 여러 개 다뤄야 할 때가 있다. 가령 한 반에 30명인 학생의 점수를 가지고 순위를 매긴다고 해보자. 우리가 지금 할 수 있는 것은 30개의 객체를 만드는 것이다.

```
int grade1;
int grade2;
int grade3;
// ...
int grade30;
```

딱 봐도 뭔가 좋지 않다. 만일 학생 수가 30명이 아니라 100명이라면? 1000명이라면? 코드를 읽기도 매우 힘들어질 것이고, 작성하는 것도 쉽지 않을 것이다. **배열**(Array)은 같은 타입으로 된 여러 개의 객체를 한번에 다루고자 할 때 사용한다.

위의 예시를 배열로 바꿔보면 아래와 같다.

```
int[] grades = new int[30];
grades[0] = 100;
grades[1] = 80;
// 후략..
```

이전보다는 데이터를 다루기가 한결 편해졌음을 확인할 수 있다. 배열의 각 문법에 대해서 좀 더 살펴보도록 하자.

문법

배열의 기본적인 문법을 확인해보자.

```
// 배열 타입의 객체 생성은 아래처럼 지정할 수 있다.
// {type_of_element}[] identifier = new {type_of_element}[array_size];
// 배열의 이름은 복수형으로 짓자.
int[] numbers = new int[30];
```

배열 타입의 객체를 생성할 때는 new 연산자를 사용한다. 위의 예시에서 arr은 int 타입의 객체 30개를 다룰 수 있는데, 객체 하나를 **원소**(Element)라고 한다. 초기화 방법은 아래와 같다.

```
int[] numbers = new int[5] { 1, 2, 3, 4, 5 };
```

혹은 아래처럼 작성할 수도 있다.

```
int[] numbers = { 1, 2, 3, 4, 5 };
```

배열의 원소는 연속적으로 메모리에 배치된다. arr을 그림으로 나타내면 아래와 같다.

데이터	1	2	3	4	5
메모리 주소	0x1234	0x1238	0x123c	0x1240	0x1244

각 원소에 접근하려면 인덱서를 사용한다. 문자열과 마찬가지로 인덱스는 0부터 시작한다.

```
int[] arr = { 1, 2, 3, 4, 5 };
arr[0]; // 배열의 첫 번째 원소. 값은 1
arr[1]; // 배열의 두 번째 원소. 값은 2
arr[2]; // 배열의 세 번째 원소. 값은 3
arr[3]; // 배열의 네 번째 원소. 값은 4
arr[4]; // 배열의 다섯 번째 원소. 값은 5
```

인덱스를 0부터 시작하는 이유

인덱스는 왜 하필 1이 아니라 0부터 시작하는 것일까? 여기엔 과학적인 이유가 숨어 있다. 이전에 배운 것을 다시 생각해보자. 메모리를 사용하기 위해선 주소가 필요하다고 했다. 마찬가지로 배열의 각 원소에 접근하기 위해서는 주소가 있어야 한다. 원소는 메모리에 연속적으로 할당되어 있고, 각 원소의 데이터 크기를 알고 있기 때문에 배열의 시작 주소만 알고 있다면 각 원소의 주소를 계산할 수 있다. 그래서 인덱스가 0부터 시작하는 것이다. 아래의 그림을 다시 보자.

데이터	1	2	3	4	5
메모리 주소	0x1234	0x1238	0x123c	0x1240	0x1244

다차원 배열

배열을 사용할 때 메모리를 논리적으로 구분하여 사용할 수 있다. 가령 학급 당 학생 수가 30명이고, 10개의 반이 있다고 해보자. 이를 일차원 배열로 나타내면 총 학생 수가 300명이므로 아래와 같이 정의할 수 있다.

```
int[] grades = new int[300];
```

이 배열을 가지고 특정 학생의 데이터에 접근하려면 약간의 계산식을 거쳐야 할 것이다.

```
// 각 원소의 데이터는 적절하게 초기화 되어 있다고 가정한다.
int[] grades = new int[300];
grades[2 * 30 + 2]; // 3반의 3번째 학생의 데이터에 접근했다.
```

보다시피 직관적이지 않다. 이런 경우 **다차원 배열***(Multidimentional Array)을 사용할 수 있다.

* 이런 의미에서 우리가 위에서 배운 배열을 일차원 배열(Single-Dimentional Array)이라고 한다.

```
// 다차원 배열은 ,를 붙여서 차원 수를 늘릴 수 있다.
int[,] grades = new int[10, 30]; // 2차원 배열
grades[2, 2]; // 계산식을 사용하지 않고 직관적으로 사용할 수 있다.

// ,를 여러 개 사용할 수 있다. 다만 4차원 이상으로는 잘 사용하지 않는 편이긴 하다.
int[,,] someData = new int[3, 4, 5];
```

초기화는 아래와 같다.

```
// 2차원 배열의 초기화
int[,] array2D = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// 초기화를 해주는 경우 자동으로 크기를 계산할 수 있기 때문에 크기를 명시하는 걸 생략할 수 있다.
int[,] array2Da = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };

// 3차원 배열의 초기화
int[,,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                { { 7, 8, 9 }, { 10, 11, 12 } } };
```

가변 배열

가변 배열(Jagged Array)은 다차원 배열과 헷갈릴 수 있지만, 각 원소가 배열 타입인 배열이라고 할 수 있다.

```
// 각 원소의 타입이 int[]고, 3개의 원소를 가지는 배열이다.
int[][] jaggedArray = new int[3][];
// 각 원소는 서로 다른 크기일 수 있다. 다시 말해 들쭉날쭉(jagged)하다.
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[4];
jaggedArray[2] = new int[] { 1, 10, 7}; // 1차원 배열 사용하는 것처럼 하면 된다.
```

더해보기

1. 백준 온라인 저지에서 아래의 문제를 풀어보세요.
    a. 1000번 - - A+B 다국어
    b. 1001번: A-B
    c. 10998번: A×B
    d. 1008번 - - A/B 스페셜 저지
    e. 10869번: 사칙연산
    f. 10926번: ??!
    g. 18108번: 1998년생인 내가 태국에서는 2541년생?!
    h. 3003번: 킹, 퀸, 룩, 비숍, 나이트, 폰
    i. 10430번: 나머지
    j. 2588번: 곱셈
2. 프로그래머스에서 아래의 문제를 풀어보세요.
    a. 코딩테스트 연습 - 주사위의 개수 | 프로그래머스 스쿨
    b. 코딩테스트 연습 - 문자열 정렬하기 (2) | 프로그래머스 스쿨
3. C#에서 30개의 `int` 타입 원소를 가지는 배열을 선언하고, `{1, 2, 3, ..., 30}`으로 초기화하는 3가지 가능한 방법을 코드로 작성하세요.
4. 배열의 인덱스가 1이 아닌 0부터 시작하는 이유를 메모리 주소 및 계산 원리와 연관지어 서술하세요.
5. 다차원 배열 (`int[,]`)과 가변 배열 (`int[][]`)의 정의를 명확히 구분하고, 이 둘의 가장 큰 차이점을 설명해 보세요.
6. C#에서 3개의 행과 4개의 열을 가지는 2차원 배열을 선언하고, `{ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} }`로 초기화하는 코드를 작성해 보세요.
7. 다음 코드가 메모리에서 동작하는 원리를 설명하고, 3반의 3번째 학생 점수(인덱스 2, 2)에 접근하는 것이 직관적이지 않은 이유를 서술해 보세요. (반 하나의 학생 수는 30명이라고 가정합니다.)

```
int[] grades = new int[300];
grades[2 * 30 + 2];
```

8. 배열은 캐싱 친화적이기 때문에 정말 중요합니다. 배열의 사용 예시 중에는 룩업 테이블(Lookup Table)이 있는데요. 룩업 테이블에 대해서 조사해보세요.

참고자료

- C# 교과서
- 왜 배열은 0부터 시작할까? 프로그래밍의 비밀