배열의 원소가 저장된 순서에 따라 접근하지 않고 인덱스 값으로 직접 접근할 수 있습니다. 하지만 이것이 그렇게 간단한 것은 아닙니다. 왜냐하면 개발자가 알고 있는 인덱스 값은 단순한 정수이지만 메모리의 주소값은 16진수이면서 프로그램이 실행되어야 결정되기 때문입니다. 개발자가 알고 있는 인덱스 값은 추상화된(컴퓨터의 내부 구조나 메모리 주소와 무관하게 개념적으로 정의된) 값이며, 메모리의 주소값은 실제 물리적인 위치값(주소값)입니다. 따라서 운영체제는 접근하려는 원소의 인덱스를 이용하여 실제 메모리 주소를 계산합니다. 배열의 이러한 특징 때문에 배열은 추상화된 의미와 구체화된 의미가 같은 자료구조라고 말할 수 있습니다.

그렇다면 '물리적인 순서(메모리의 주소값)'와 '순서(추상화된 인덱스 값)'는 어떤 관계일까요? 추상화된 인덱스 값, 혹은 순서는 제1장에서 배운 '추상화된 값'입니다. 배열의 인덱스 값은 물리적으로 컴퓨터가 처리하는 값이 아니라 사람이 이해하고 직관적으로 받아들이기 위해 정의된 값입니다. 그리고 그렇게 추상화된 값을 컴퓨터가 해석하여 물리적인 값으로 변환하고, 이를 전기적인 신호로 바꾸어 처리하는 것입니다.

[그림 2-1]을 보면서 설명하겠습니다. 배열의 물리적인 저장 순서는 배열의 인덱스에 의해서 결정되며, 그 순서가 바로 메인 메모리에서 저장 위치의 순서가 됩니다.

[그림 2-1]에서 왼쪽의 16진수가 메모리의 주소입니다. 실제로 우리가 사용하고 있는 메모리의 각 영역에는 주소가 부여되어 있습니다. 그리고 상자 안에 있는 숫자는 원소값(데이터)입니다. 우리가 프로그램을 작성하고 실행시키면 16진

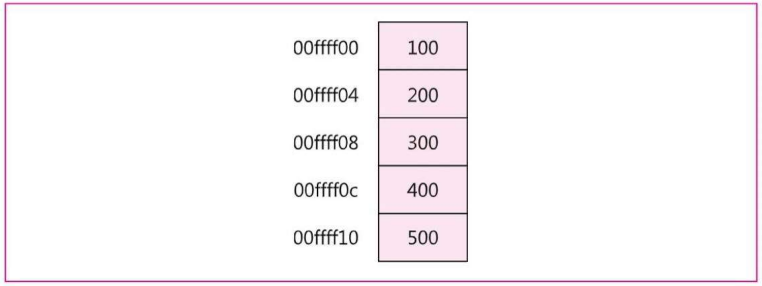

그림 2-1 메모리 할당과 메모리 주소

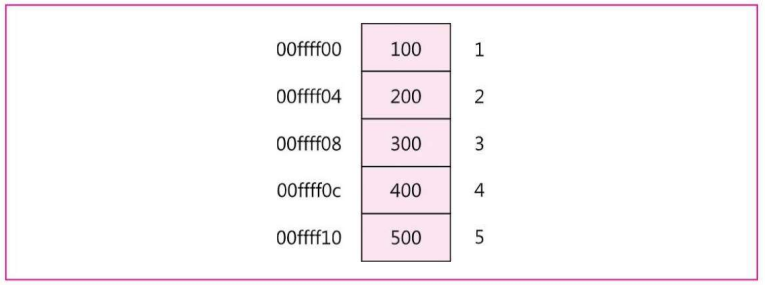

그림 2-2 메모리 할당과 인덱스 값의 관계

수 값의 주소를 찾아내서 데이터를 찾아오고 데이터를 저장하게 되는 것입니다. [그림 2-2]는 [그림 2-1]에 '인덱스 값'을 붙여 놓은 것입니다.

즉, 오른쪽은 추상화된 인덱스 값이 되며 왼쪽은 실제 메인 메모리의 주소값이 됩니다. 그렇다면 이러한 주소 혹은 배열의 인덱스 값이 중요한 이유는 무엇일까요?

[그림 2-3]과 같이 컴퓨터는 운영체제의 지휘 아래서 메모리의 물리적인 주소값을 전달받아 자료를 가져오고, CPU에서 가져온 자료를 계산하게 됩니다. 그리고 개발자는 프로그램에서 선언한 배열과 인덱스 값을 이용하여 알고리즘을 작성하고 그 알고리즘에 따라 프로그램을 완성합니다. 운영체제는 개발자의 추상화된 값과 컴퓨터의 물리적인 값을 연결시켜 줍니다. 그리고 프로그래밍 언어는 개발자와 운영체제 간의 의사소통을 위한 도구라고 생각하면 됩니다.

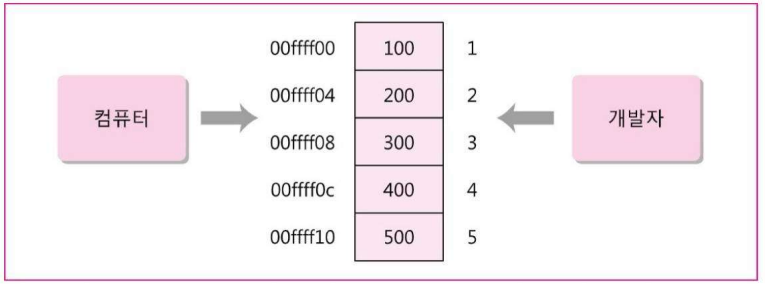

그림 2-3 메모리 영역의 추상화와 구체화