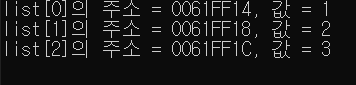
배열에 따른 선형리스트의 구현

선형 리스트란 리스트에 나열한 데이터들이 일정한 순서를 가지고 있는 것을 말한다. 순차 리스트라고도 부른다. 선형리스트의 중요한 특징은 원소들 간의 논리적인 순서와 메모리에 저장되는 물리적인 순서가 같다는 것이다. 따라서 메모리에 데이터가 연속적으로, 순서대로 저장이 되기 때문에, 데이터의 시작 위치와, 원소의 크기를 가지고 특정 원소의 위치를 알 수 있다는 점이다. 가장 쉽게 선형 리스트를 표현하기 위해 배열을 사용할 수 있다. 배열은 인덱스, 데이터로 구성되어 있다. 선형 리스트의 특징대로 선형 리스트에 데이터를 저장하면, 메모리 상에 저장될 때에도 데이터가 나열된 순서대로 저장이 된다. 32bit 컴파일 환경에서 만약 [그림 1]처럼 int 배열 list에 1, 2, 3을 순서대로 저장했다고 하자.



[그림 1. list 배열]

list 배열에 1, 2, 3이 논리적인 순서대로 저장이 되었으므로 실제 메모리상에도 논리적인 순서와 같이 차례대로 저장되었을 것이다.



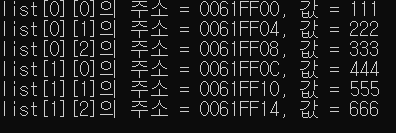
[그림 2. 리스트의 주소와 값]

[그림 2]에서 확인해 본 결과, list의 주소는 각각 4씩 증가하였고 그 값은 차례로 1, 2, 3 이었다. 이를 보아 32bit 환경에서 list에 정보를 저장할 때 메모리가 자료형의 크기(int, 4bytes)씩 증가하여 순서대로 저장됨을 알 수 있다. 앞에 기술한 선형 리스트의 특징에 따라 만약 메모리 상에서 list[0]의 위치와 list의 구성 자료형을 알면 list[2]의 값 또한 알아 낼 수 있을 것이다. 앞의 예시들은 선형리스트를 1차원 배열로 표시하였다. 하지만 배열의 원소가 또 다른 배열인 2차원 배열로도 순차 리스트를 표현할 수 있다.



[그림 3. 2차원 list 배열]

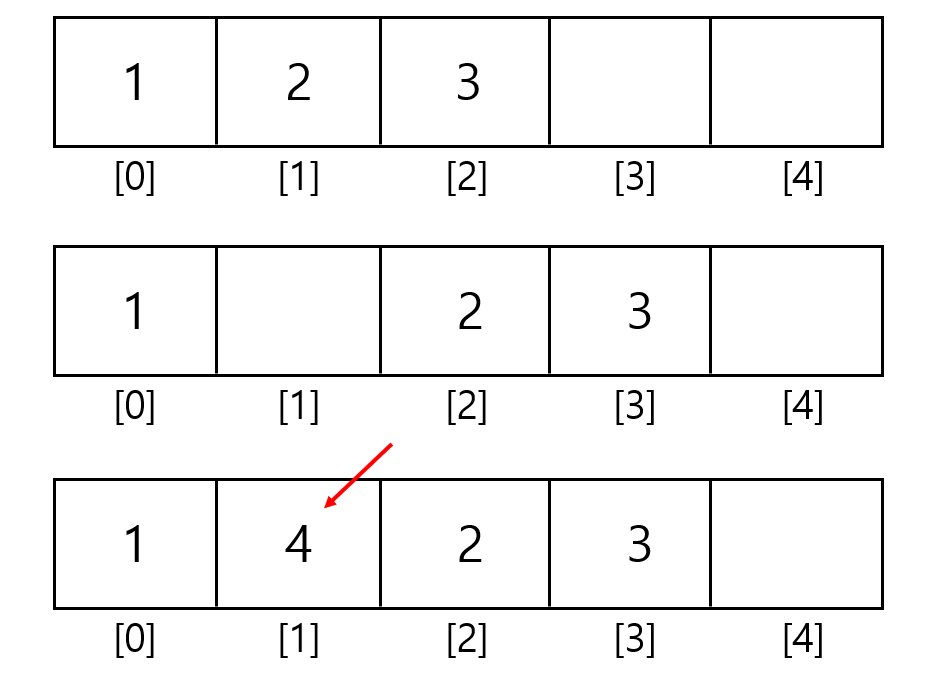
[그림 3]과 같이 list를 2차원 배열로 선언하여 그 원소를 또 다른 배열인 {111, 222, 333}과, {444, 555, 666}으로 선언하고 초기화 하였다고 하자. 위의 1차원 배열과 마찬가지로 각 원소들의 주소와 값들을 출력해보자.



[그림 4. 2차원 list의 주소와 값]

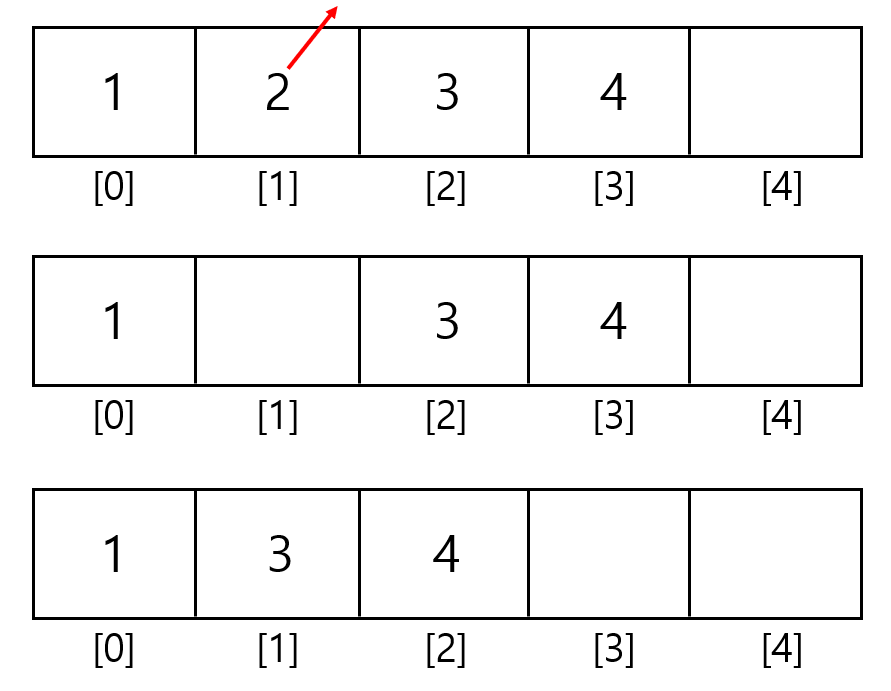
[그림 4와] 같이 2차원 배열로 구현한 선형 리스트 또한 실제 메모리에서 구성 요소 int의 크기만큼 주소값이 증가하였고, 그에 따른 값도 일치하였다. 이를 통해 다차원 배열로도 선형 리스트를 구성할 수 있고, 확장할 수 있음을 알 수 있다.

앞에서는 배열로 어떻게 선형 리스트를 표현하는지를 알아보았다. 만약 배열로 구성한 선형 리스트에 새로운 데이터를 추가하려고 하면 어떻게 해야 하는지 알아보자. 만약 선형 리스트의 맨 마지막에 데이터를 삽입해야 한다면, 추가적인 작업 없이 그냥 맨 마지막 공간에 데이터를 삽입하면 될 것이다. 하지만 리스트의 중간에 데이터를 삽입하고자 하면, 추가적인 작업이 필요하다. [그림 5]의 예시와 같이 데이터 2, 3을 오른쪽으로 한 칸 밀고, 그 사이에 4라는 데이터를 추가하는 작업이 필요하다.



[그림 5. 선형 리스트에서의 데이터 삽입]

데이터 삭제 시에도 비슷한 작업이 필요하다. 만약 삭제할 데이터가 리스트의 맨 마지막 원소라면 그냥 데이터를 삭제해도 선형 리스트의 성질이 유지된다. 하지만 삭제할 데이터가 리스트의 중간에 있는 원소라면 선형 리스트의 성질을 유지하기 위해, 모든 데이터를 한 칸씩 앞으로 당기는 작업이 필요하다. [그림 6]의 예시와 같이 만약 데이터 2를 삭제하고 싶으면, 먼저 데이터를 삭제하고, 선형 리스트의 성질을 만족시키기 위해서 뒤쪽에 있는 데이터를 모두 한 칸씩 앞으로 당겨야 한다.



[그림 6. 선형 리스트에서의 데이터 삭제]

앞에서 본 선형리스트의 구현, 삽입, 삭제와 같은 특성들은 선형 리스트의 장점과 단점을 보여준다. 먼저 배열을 사용하여 구현이 쉽다는 장점이 있다. 하지만 삽입, 삭제 작업시에 최악의 경우 거의 모든 데이터를 한 칸씩 밀거나 당기는 작업이 필요하여 많은 overhead가 발생한다는 단점이 있다.