

배열 - 메모리 상에 원소를 연속하게 배치한 자료구조
자료구조의 배열에서는 길이 변경 가능

1. $O(1)$ 에 k번째 원소를 확인/변경 가능
2. 추가적으로 소모되는 메모리의 양(=overhead)가 거의 없음
3. Cache hit rate가 높음
4. 메모리 상에 연속한 구간을 잡아야 해서 할당에 제약이 걸림

* 3) Cache Hit Rate?

Cache Hit이란 CPU가 참조하고자 하는 메모리가 캐시에 존재하고 있는 경우를 말한다.

이 비율이 높을수록 좋은 성능을 가질 수 있다.

우선 메모리에 대한 개념 중 참조 지역성 원리라는 것이 있다.

참조 지역성 원리란 동일한 값 또는 해당 값에 관계된 스토리지 위치가 자주 액세스되는 특성으로, 지역성의 원리(Principle of Locality)라고도 부른다.

이 참조 지역성에는 3가지 종류가 있다.

1. 공간 지역성(Spatial Locality) : 참조된 주소와 인접한 주소의 내용이 다시 참조되는 특성
2. 시간 지역성(Temporal Locality) : 최근에 참조된 주소는 빠른 시간 내에 다시 참조되는 특성
3. 순차 지역성(Sequential Locality) : 데이터가 순차적으로 액세스 되는 특성, 공간 지역성에 편입되어 설명되기도 함

배열은 메모리 상 연속적으로 데이터가 저장되어있다고 했다.

즉, 배열은 공간 지역성이 좋아 높은 Cache Hit Rate를 가진다고 할 수 있다.

임의의 위치에 있는 원소를 확인/변경 = $O(1)$

원소를 끝에 추가 = $O(1)$

마지막 원소를 제거 = $O(1)$

임의의 위치에 원소를 추가/임의 위치의 원소 제거 = $O(N)$

배열 초기화

```
01 int a[21];  
02 int b[21][21];  
03
```

```

04 // 1. memset (0, -1 아니면 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31)
05 memset(a, 0, sizeof a);
06 memset(b, 0, sizeof b);
07
08 // 2. for
09 for(int i = 0; i < 21; i++)
10     a[i] = 0;
11 for(int i = 0; i < 21; i++)
12     for(int j = 0; j < 21; j++)
13         b[i][j] = 0;
14
15 // 3. fill (초기화)
16 fill(a, a+21, 0);
17 for(int i = 0; i < 21; i++)
18     fill(b[i], b[i]+21, 0);

```

STL vector

배열과 비슷함

크기 조정이 가능

```

01 #include <bits/stdc++.h>
02 using namespace std;
03
04 int main(void) {
05     vector<int> v1(3, 5); // {5,5,5};
06     cout << v1.size() << '\n'; // 3
07     v1.push_back(7); // {5,5,5,7};
08
09     vector<int> v2(2); // {0,0};
10     v2.insert(v2.begin()+1, 3); // {0,3,0};
11
12     vector<int> v3 = {1,2,3,4}; // {1,2,3,4}
13     v3.erase(v3.begin()+2); // {1,2,4};
14
15     vector<int> v4; // {}
16     v4 = v3; // {1,2,4} deep copy
17     cout << v4[0] << v4[1] << v4[2] << '\n'; // 1 2 4
18     v4.pop_back(); // {1,2}
19     v4.clear(); // {}
20 }

```

$O(1)$ (push_back, pop_back, clear)
 $O(N)$ (insert, erase)
 v3에는 영향 X

원소가 메모리에 연속하게 저장되어있어 각 원소 접근: $O(1)$

* push-front, pop-front ... $O(N)$

```
01 vector<int> v1 = {1,2,3,4,5,6};
02
03 // 1. range-based for loop (since C++11)
04 for(int e : v1)      e에 원소가 하나씩 들어감(복사됨)
05     cout << e << ' ';      for(int& e:v1) ... (원본)
06
07 // 2. not bad
08 for(int i = 0; i < v1.size(); i++)
09     cout << v1[i] << ' ';
10
11 // 3. ***WRONG***      unsigned int여서 -1하면
12 for(int i = 0; i <= v1.size()-1; i++)      값이 이상하게 됨
13     cout << v1[i] << ' ';
```

