Array

정의

배열은, 메모리 상에 원소를 연속하게 배치한 자료구조로, 원래의 배열에서는 배열의 길이를 변경하는 게 불가능하지만 자료구조상의 배열은 길이의 변경이 자유롭다고 생각하고 작성

성질
- 추가적으로 소모되는 메모리 양(overhead) 가 거의 없음
-*Cache hit rate가 높음
- 메모리 상에 연속한 구간을 잡아야 해서 할당에 제약이 걸림

시간 복잡도 (Time complexity)
- 임의의 위치에 있는 원소를 확인/변경 = O(1)
- 원소를 끝에 추가 = O(1)
- 마지막 원소를 제거 = O(1)
- 임의의 위치에 원소를 추가/제거 = O(N)

- Cache hit rate(ratio)

Cache Hit Ratio란 말 그대로 캐시가 적중되는 정도를 뜻합니다. 해당 적중률이 높을수록 CPU와 주기억장치의 속도 차이로 인한 병목현상을 최소화할 수 있습니다.
이러한 Cache Hit Ratio는 캐시의 Locality, 즉 지역성에 의해 높아집니다.
지역성이란 CPU의 주기억장치의 특정 부분에 위치한 코드나 데이터에 빈번히 혹은 집중적으로 액세스 하게 되는 현상을 말합니다.
지역성에는,
시간적 지역성 / 공간적 지역성 / 순차적 지역성이 있음

시간적 지역성 : 최근에 액세스 된 프로그램이나 데이터가 가까운 미래에 다시 액세스 될 가능성이 높음을 의미합니다. 반복루프, 서브루틴 호출, 공통 변수가 대표적인 예시입니다.

공간적 지역성 : 기억장치 내에 인접하여 저장된 데이터들이 연속적으로 액세스될 가능성이 높음을 의미합니다. 표, 배열의 데이터가 그 대표적 예입니다.

순차적 지역성 : 분기가 발생하지 않는 이상 명령어들이 기억장치에 저장된 순서대로 인출되어 실행됨을 의미합니다. 이는 공간적 지역성에 편입되어 설명되기도 합니다.

Array

예시코드

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
void insert(int idx, int num, int arr[], int& len);
void erase(int idx, int arr[], int& len);
void printArr(int arr[], int& len);

void insert(int idx, int num, int arr[], int& len) {
    for(int i = len; i > idx; i--){
        arr[i] = arr[i - 1];
    }
    arr[idx] = num;
    len++;
}

void erase(int idx, int arr[], int& len) {
    len--;
    for (int i = idx; i < len; i++) {
        arr[i] = arr[i + 1];
    }
}

void printArr(int arr[], int& len) {
    for (int i = 0; i < len; i++) cout << arr[i] << ' ';
    cout << "\n\n";
}

int main() {
    int arr[10] = { 1,2,3,4,5,6,7 };
    int len = 7;
    insert(3, 9, arr, len);
    erase(4, arr, len);
```

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
int a[21]; int b[21][21];

// 전체를 특정 값으로 초기화
// 전역선언 할 경우 처음부터 0으로 초기화 되있긴함

// for
int fr(int len) {
    for (int i = 0; i < len; i++) {
        a[i] = 0;
    }
    for (int i = 0; i < len; i++) {
        for (int j = 0; j < len; j++) {
            b[i][j] = 0;
        }
    }

}

// STL algorithm header -> fill

int fl(int len) {
    fill(a, a + len, 0);
    for (int i = 0; i < len; i++) {
        fill(b[i], b[i] + len, 0);
    }
}
```

Array

STL

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main(void) {
  vector<int> v1(3, 5); // {5,5,5};
  cout << v1.size() << '\n'; // 3
  v1.push_back(7); // {5,5,5,7};

  vector<int> v2(2); // {0,0};
  v2.insert(v2.begin()+1, 3); // {0,3,0};

  vector<int> v3 = {1,2,3,4}; // {1,2,3,4}
  v3.erase(v3.begin()+2); // {1,2,4};

  vector<int> v4; // {}
  v4 = v3; // {1,2,4}
  cout << v4[0] << v4[1] << v4[2] << '\n';
  v4.pop_back(); // {1,2}
  v4.clear(); // {}
}
```

추가 및 삭제

assign([개수],[요소]) - 벡터에 개수만큼 요소로 할당시킵니다.
push_back([요소]) - 벡터 Back에 원소를 추가합니다.
pop_back() - 벡터 Back에 원소를 삭제합니다.
insert([위치], [요소]) - 벡터 해당 위치에 요소를 추가합니다.
erase([위치]) - 벡터 해당 위치에 원소를 삭제합니다.
clear() - 벡터 전체를 초기화합니다.

해당 값 표시

[위치] - 벡터는 배열과 같이 vec[i] 시 i 번째의 원소 값을 표시할 수 있습니다.
at([위치]) - 벡터 해당 위치에 요소를 표시합니다.
front() - 벡터 맨 앞의 요소를 표시합니다.
back() - 벡터 맨 뒤의 요소를 표시합니다.

반복자

begin() - 맨 앞 반복자를 반환합니다.
end() - 맨 뒤 반복자를 반환합니다.

기타

size() - 벡터 전체의 크기를 반환합니다.
resize([개수]) - 벡터 크기를 개수만큼 초기화합니다.
empty() - 벡터가 비어있는지 아닌지 확인합니다.
swap() - 두 벡터의 내용을 바꿉니다.