# STL Containers

# STL이란?

- Standard Template Library
  - 프로그램에 필요한 자료구조와 알고리즘을 **template으로 제공하는 라이브러리**
  - 반복자(iterator), 컨테이너(container), 알고리즘(algorithm)

- 컨테이너
  - **데이터를 저장하고 관리**하기 위한 클래스

- 반복자
  - STL 컨테이너에 **저장된 원소들을 가리키는 포인터**
  - 컨테이너에 사용 되는 반복문

- 알고리즘
  - STL에서 제공하는 **함수**

# STL Containers

- 자주 쓰는 컨테이너

- **arry** : 배열
= 적은 양의 자료에 유리
= **크기 변경 불가**

- **vector** : 가변 배열
= 적은 양의 자료에 유리
= **크기 변경이 가능**

- **list** : 양방향 연결 리스트
= 적은 양의 자료에 유리
= 크기 변경 가능
= **중간 삽입 삭제가 가능**

- **map** : 이진 탐색 트리(균형 이진 트리 / 레드-블랙 트리)
= **key와 value**를 가지며 따로 저장
= **많은 양의 자료에 유리**
= **적은 양에는** 오버헤드로 **손해**

# STL Containers

- 컨테이너 사용법

- **arry** : #include <array>

```
std::array<데이터 형, 크기> _array;
_array[index] = value;
, _array.at(index) = value;
```

- **vector** : #include <vector>

```
std::vector<데이터 형> _array;
```

array와 사용 방법 동일

```
_array.clear();
```

- **list** : #include <list>

```
std::list<데이터 형> _list;
_list.push_back(value);
_list.remove(value);
_list.clear();
```

# STL Containers

- 컨테이너 사용법

- **map** : #include <map>

```
std::map<키 데이터 형, 데이터 형> _map;

_map.insert(std::pair<키 데이터 형, 데이터 형>(key, value));
, _map[key] = value;

std::map<키 데이터 형, 데이터 형>::iterator iter = _map.find(key);
iter->first : key
iter->second : value

_map.erase(key);
_map.clear();

if (_map.end() == _map.find(key))
{
    // 해당 키 값을 가진 데이터가 없다.
}
```

# STL Containers

- 반복문

```
for(const auto& element : _arry)
{
    element
}

for(const auto& element : _list)
{
    element
}

for(const auto& element : _map)
{
    element. first
    element. second
}
```

# STL Containers

- 반복문

```
std::map<키 데이터 형, 데이터 형>::iterator iter;
for (iter = _map.begin(); _map.end() != iter; iter++)
{
    iter->first;
    iter->second;
}
```