

STL

(Standard Template Library)

Data Structures
C++ for C Coders

한동대학교 김영섭 교수
idebtor@gmail.com

vector, list, deque, queue, stack, set
map

Containers

- A container is a way to store data, either built-in data types like int and float, or class objects
- The STL provides several basic kinds of containers
 - **<vector>** : one-dimensional array
 - **<list>** : double linked list
 - **<deque>** : double-ended queue
 - **<queue>** : queue
 - **<stack>** : stack
 - **<set>** : set
 - **<map>** : associative array

Containers

STL 컨테이너	특 징
vector	<ul style="list-style-type: none"> - 동적 배열이므로 배열의 크기를 변경할 수 있다. - 임의 접근이 가능하며, 뒤에서의 삽입이 빠르다.
list	<ul style="list-style-type: none"> - 연결 리스트이므로 데이터를 순차적으로 접근하고 관리할 때 유용하다. - 위치에 상관없이 삽입과 삭제가 빠르다.
deque	<ul style="list-style-type: none"> - 데크라고 한다. - 임의 접근이 가능하며, 앞과 뒤에서의 삽입이 빠르다.
map	<ul style="list-style-type: none"> - 특정 키(key)에 의해서 데이터를 접근하고 관리할 수 있다 - 키를 통해 값을 접근하며, 삽입과 삭제가 빠르다.
set	<ul style="list-style-type: none"> - 원소들을 순서대로 관리하며, 소속 검사와 삽입, 삭제가 빠르다. - 중복된 원소를 허용하지 않는다.
stack	<ul style="list-style-type: none"> - top에서만 삽입과 삭제가 가능하다. - LIFO(Last In First Out) 방식으로 데이터를 삽입, 삭제 한다.
queue	<ul style="list-style-type: none"> - 삽입은 뒤쪽에서, 삭제는 앞쪽에서 수행한다. - FIFO(First In First Out) 방식으로 데이터를 삽입, 삭제 한다.

Sequence 순차 Containers

Associative 연관 Containers

Adaptor Containers

C++ containers, sets, maps

- **Container:**
 - data type for operating on a group of elements Example: array
- **Sets:**
 - container for distinct, ordered data stored in a balanced binary tree structure supporting fast search
- **Maps:**
 - associative container for $\langle \text{key}, \text{value} \rangle$ pairs (where keys are distinct and ordered), stored in a balanced binary tree structure

Supported operations on sets, maps

- Add and remove element (if not already in the container)
- Get count of elements
- Check membership
- **Addition, removal, search guaranteed to take $O(\log N)$ time**

Set example

```
set<int> s;

for(int i = 1; i <= 100; i++)          // insert 100 elements, [1..100]
    s.insert(i);

s.insert(42);                          // does nothing, 42 already exists in set

for(int i = 2; i <= 100; i += 2)       // Erase even values
    s.erase(i);

int n = int(s.size());                 // n will be 50

if (s.find(42) != s.end())
    cout << "42 is in set" << endl;
else
    cout << "42 is not in set" << endl;
```

Map example

```
#include <string>
#include <iostream>
#include <map>
#include <utility>
using namespace std;
int main() {
    map<int, string> team;

    // note the use of array index notation
    team[52] = "Mike C.";
    team[19] = "David D.";
    team[75] = "John A.";
    team[53] = "Peter Q.";

    cout << "team[53]=" << team[53] << endl;
    cout << "size: " << team.size() << endl;
```

Map example

```
cout << endl << "Natural Order:" << endl;
for (map<int,string>::iterator it=team.begin(); it!=team.end(); ++it)
    cout << (*it).first << ": " << (*it).second << endl;

for (auto it=team.begin(); it!=team.end(); ++it)
    cout << it->first << ": " << it->second << endl;

cout << endl << "Reverse Order:" << endl;
for (auto it=team.rbegin(); it!=team.rend(); ++it)
    cout << it->first << ": " << it->second << endl;

cout << endl << "Auto Order:" << endl;
for (auto x: team)
    cout << x.first << ": " << x.second << endl;
return 0;
}
```


Map example

```
#include <string>
#include <iostream>
#include <map>
#include <utility>
using namespace std;

int main() {
    map<string, int> team;
    team["Mike C."] = 52;                // using array index notation
    team["John M."] = 33;
    team.insert(pair<string, int>("David D.", 19)); // using insert() and pair struct
    team.insert(make_pair("Peter Q.", 53));        // using insert() and "make_pair()"

    cout << "size: " << team.size() << endl;
    for (auto it = team.begin(); it != team.end(); ++it)
        cout << it->first << ": " << it->second << endl;
}
```

Pair

- Stores a pair of objects, first of type T1, and second of type T2.
- ```
struct pair<T1, T2>
{
 T1 first;
 T2 second;
};
```

■

# Pair

- Stores a pair of objects, first of type T1, and second of type T2.

- ```
struct pair<T1, T2>
{
    T1 first;
    T2 second;
};
```

```
5  #include <iostream>
6  #include <map>
7  using namespace std;
8
9  int main() {
10     // using initialization, pair<> construct, and make_pair() function
11     map<char, int> chart { pair<char,int>('A', 65),
12                           pair<char,int>('C', 67),
13                           make_pair('D', 68),
14                           make_pair('B', 66) };
15
16     for (auto item: chart) {
17         cout << "ascii: " << item.first << "\t";
18         cout << " code: " << item.second << endl;
19     }
20     cout << chart['B'] << endl;
21     return 0;
22 }
```



Summary

&

quaestio quaestio qo → q ? ? ?