#0 — STL을 이용한 문제 해결

Solving Problems with C++ STL

by

Suhyun Park

"**바퀴**를 재발명하지 마라"

Standard **T**emplate **L**ibrary

- ✓ C++ 표준
- ✓ 언어에서 기본적으로 제공하는 템플릿 라이브러리
- ✓ 개발에 자주 사용되는 여러 자료 구조, 알고리즘 등이 미리 구현되어 있음
  - 큐, 스택, 덱, 힙, 동적 배열, 집합 등의 자료 구조
  - 최댓값, 최솟값, 이분 탐색, 엄청 빠른 정렬 등의 알고리즘

진짜 그냥 갖다 써도 돼요?

- ✓ 거의 **모든** 대회와 코딩 테스트에서 사용 가능!
- ✓ 단 여기서는 쓸 수 없다
  - 서강대학교 C프로그래밍 / 컴실1 실기시험
  - 삼성 역량 테스트 B형, C형

STL

그냥 갖다 써 봅시다

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {
    queue<int> q;

    for (int i = 0; i < 10; i++) q.emplace(i);

    while (!q.empty()) {
        cout << q.front() << '\n';
        q.pop();
    }

    return 0;
}
```

```cpp
#include <iostream>
#include <queue>

using namespace std;
```

각각의 자료구조는 각각의 **헤더**에 선언되어 있다

- ✓ 적절한 헤더를 `include` 해서 사용

```
6  int main() {
7      queue<int> q;
8
9      for (int i = 0; i < 10; i++) {
10         q.emplace(i);
11     }
```

Standard **“Template”** Library

- ✓ STL의 (거의) 모든 자료구조와 알고리즘은 **템플릿**
- ✓ `queue<int>` 하면 `int`의 큐이고, `queue<string>` 하면 `string`의 큐이고···

PS에서 자주 쓰이는 STL 알고리즘

- ✓ `sort` – **엄청빠른**정렬
- ✓ `min`, `max`, `minmax` – 다 아시죠?
- ✓ `lower_bound`, `upper_bound` – 이분 탐색
- ✓ `next_permutation` - 1234 1243 1324 1342 1423 · · ·

이외에도 많음!

PS에서 자주 쓰이는 STL 자료 구조

- ✓ `string` – 문자열
- ✓ `vector` – 동적 배열
- ✓ `queue`, `stack` – 큐, 스택
- ✓ `priority_queue` – 우선순위 큐 (힙)

이외에도 `deque`, `set`, `multiset`, `map`, `pair`, `tuple`, ···
외우려고 하기보단 짜면서 익숙해지는 게 좋다

참고하면 좋은 사이트

- ✓ **C++ reference** cppreference.com
- ✓ **Standard C++ Library reference** cplusplus.com/reference

오늘은 중급에서 다루는 것들 중 핵심적인 것들만 알아봅니다

`std::sort(begin, end)`

defined in header `<algorithm>`

- ✓ **엄청빠른**정렬
- ✓ 퀵소트를 직접 구현할 필요가 없다

sort – 정렬

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int a[] = {1, 7, -6, 4, -3};

int main() {
    sort(a, a + 5);
    for (int i = 0; i < 5; i++) {
        cout << a[i] << ' ';
    } // -6 -3 1 4 7
    return 0;
}
```

```
std::sort(begin, end, compare)
```

- ✓ 보통 작은 게 먼저 오지만, 큰 게 먼저 오게 할 수도 있다
- ✓ 아니면 함수를 짜서 정렬 순서를 직접 결정할 수도 있다

sort – 정렬

```cpp
#include <iostream>
#include <algorithm>
#include <functional>

using namespace std;

int a[] = {1, 7, -6, 4, -3};

int main() {
    sort(a, a + 5, greater<int>());
    for (int i = 0; i < 5; i++) {
        cout << a[i] << ' ';
    } // 7 4 1 -3 -6
    return 0;
}
```

sort – 정렬

```cpp
#include <iostream>
#include <algorithm>
#include <cmath>

using namespace std;

int a[] = {1, 7, -6, 4, -3};

bool comp(int a, int b) {
    return abs(a) < abs(b);
}

int main() {
    sort(a, a + 5, comp);
    for (int i = 0; i < 5; i++) {
        cout << a[i] << ' ';
    } // 1 -3 4 -6 7
    return 0;
}
```

sort – 정렬

```
9  bool comp(int a, int b) {
10     return abs(a) < abs(b);
11 }
```

comp(a, b)가 true라면 a가 b 앞에 옴!

sort – 정렬

```
6  struct point { int x, y; };
7
8  bool comp(const point& a, const point& b) {
9      if (a.x != b.x) return a.x < b.x;
10     return a.y < b.y;
11 }
```

```
14 sort(points, points + n, comp);
```

sort – 정렬

```
6  struct point {
7      int x, y;
8      point(int x, int y) : x(x), y(y) {};
9      bool operator<(const point& rhs) const {
10         if (x != rhs.x) return x < rhs.x;
11         return y < rhs.y;
12     }
13 };
```

```
16 sort(points, points + n);
```

`std::string`

defined in header `<string>`

char*와 string이 뭐가 다르길래?

	char*	string
입출력 메소드	scanf, printf	cin, cout
함수 사용 예	strlen(str)	str.length()
메모리	프로그래머가 직접 관리	**STL이 알아서 관리**
길이 쿼리 복잡도	$\mathcal{O}(n)$	$\mathcal{O}(\mathbf{1})$

string의 장점

- ✓ 메모리 관리를 STL이 알아서 다 해 준다 – 문제풀이에 더 집중할 수 있다
- ✓ 함수 사용이 직관적이다 (`strlen(str)` vs. `str.length()`)
- ✓ 길이 쿼리가 상수 시간이다 (전처리되어 있음)
- ✓ 어차피 `cin`, `cout` 쓰려면 `std::string` 쓰는 게···

자주 쓰는 메서드

- ✓ `string.length()`: 문자열의 길이 반환

string – 문자열

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    string str = "Hello World!";
    cout << str << '\n';
    cout << str.length() << '\n'; // 12
    return 0;
}
```

`std::vector<T>`

defined in header `<vector>`

- ✓ 크기를 알아서 관리해 주는 동적 배열
- ✓ 이제 힘들게 `malloc`, `realloc` 할 필요 없다

장단점이 좀 있는데

- ✓ 메모리 관리를 직접 할 필요가 없다 (장점)
- ✓ 원하는 크기 원하는 값으로 초기화하기 편하다 (장점)
- ✓ 일반적으로 정적 배열보다 느리다 (단점)

따라서 동적 배열이 꼭 필요할 경우나 공간을 아껴야 하는 경우에만 쓰고, 아니면 최대한 정적 배열을 쓰는 것이 좋다

초기화하기

- ✓ `vector<int> v`: 크기 0인 `int` 배열
- ✓ `vector<int> v(n)`: 크기 n인 `int` 배열
- ✓ `vector<int> v(n, -1)`: 크기 n이고 초기값 -1인 배열
- ✓ `vector<vector<int» M(n, vector<int>(m))`: $M_{n \times m}$

자주 쓰는 메서드

- ✓ `vector.size()`: 크기 반환
- ✓ `vector.resize(n)`: 크기를 n으로 설정
- ✓ `vector.emplace_back(v)`: 맨 뒤에 원소 v 추가
- ✓ `vector.pop_back()`: 맨 뒤 원소 제거

`vector.emplace_front()` 등은 $\mathcal{O}(n)$이라서 잘 사용하지 않는다

vector – 동적 배열

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> v(5);
    for (int i = 1; i <= 5; i++) {
        v.emplace_back(i);
    }
    v.resize(12);
    for (int x : v) {
        cout << x << ' ';
    } // 0 0 0 0 0 1 2 3 4 5 0 0
    return 0;
}
```

vector – 동적 배열

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    vector<int> v(5);
    for (int i = 1; i <= 5; i++) {
        v.emplace_back(i);
    }
    v.resize(12);
    sort(v.begin(), v.end());
    for (int x : v) {
        cout << x << ' ';
    } // 0 0 0 0 0 0 0 1 2 3 4 5
    return 0;
}
```

`std::queue<T>`

defined in header `<queue>`

`std::stack<T>`

defined in header `<stack>`

- ✓ **큐** – 먼저 들어간 원소가 **먼저** 나오는 자료 구조
- ✓ **스택** – 먼저 들어간 원소가 **나중에** 나오는 자료 구조
- ✓ 역시 메모리 관리를 알아서 해 준다
- ✓ 이제 힘들게 링크드 리스트를 직접 구현할 필요가 없다

초기화하기

- ✓ `queue<int> q`: 비어 있는 `int` 큐
- ✓ `stack<int> s`: 비어 있는 `int` 스택

자주 쓰는 메서드 (큐)

- ✓ `queue.empty()`: 큐가 비어 있는지 여부 반환
- ✓ `queue.size()`: 크기 반환
- ✓ `queue.emplace(v)`: 맨 뒤에 원소 v 삽입
- ✓ `queue.pop()`: 맨 앞 원소 제거
- ✓ `queue.front()`: 맨 앞의 원소 반환

자주 쓰는 메서드 (스택)

- ✓ `stack.empty()`: 스택이 비어 있는지 여부 반환
- ✓ `stack.size()`: 크기 반환
- ✓ `stack.emplace(v)`: 맨 위에 원소 v 삽입
- ✓ `stack.pop()`: 맨 위 원소 제거
- ✓ `stack.top()`: 맨 위의 원소 반환

queue, stack – 큐, 스택

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {
    queue<int> q;
    for (int i = 1; i <= 5; i++) {
        q.emplace(i);
    }
    while (!q.empty()) {
        cout << q.front() << ' ';
        q.pop();
    } // 1 2 3 4 5
    return 0;
}
```

queue, stack – 큐, 스택

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> s;
    for (int i = 1; i <= 5; i++) {
        s.emplace(i);
    }
    while (!s.empty()) {
        cout << s.top() << ' ';
        s.pop();
    } // 5 4 3 2 1
    return 0;
}
```

`std::priority_queue<T>`

defined in header <queue>

- ✓ **힙**이라고도 한다
- ✓ 들어가는 순서와 상관없이 **가장 큰** 원소가 나오는 자료구조

초기화하기

- ✓ `priority_queue<int> pq`: 비어 있는 `int` 우선순위 큐
- ✓ `priority_queue<int, vector<int>, greater<int>> pq`: 작은 게 먼저 나온다

자주 쓰는 메서드

- ✓ `priority_queue.empty()`: 큐가 비어 있는지 여부 반환
- ✓ `priority_queue.size()`: 크기 반환
- ✓ `priority_queue.emplace(v)`: 원소 v 삽입
- ✓ `priority_queue.pop()`: 가장 큰 원소 제거
- ✓ `priority_queue.top()`: 가장 큰 원소 반환

priority_queue – 우선순위 큐

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {
    priority_queue<int> pq;
    pq.emplace(-1);
    pq.emplace(4);
    pq.emplace(2);
    pq.emplace(10);
    pq.emplace(-5);
    while (!pq.empty()) {
        cout << pq.top() << ' ';
        s.pop();
    } // 10 4 2 -1 -5
    return 0;
}
```

priority_queue – 우선순위 큐

```cpp
#include <iostream>
#include <queue>
#include <vector>
#include <functional>

using namespace std;

int main() {
    priority_queue<int, vector<int>, greater<int>> pq;
    pq.emplace(-1);
    pq.emplace(4);
    pq.emplace(2);
    pq.emplace(10);
    pq.emplace(-5);
    while (!pq.empty()) {
        cout << pq.top() << ' ';
        s.pop();
    } // -5 -1 2 4 10
    return 0;
}
```

priority_queue – 우선순위 큐

```
7  struct _compare {
8      bool operator()(int a, int b) {
9          return abs(a) < abs(b);
10     }
11 };
12
13 int main() {
14     priority_queue<int, vector<int>, _compare> pq;
15     pq.emplace(-1);
16     pq.emplace(4);
17     pq.emplace(2);
18     pq.emplace(10);
19     pq.emplace(-5);
20     while (!pq.empty()) {
21         cout << pq.top() << ' ';
22         s.pop();
23     } // 10 -5 4 2 -1
24     return 0;
25 }
```

✓ **정렬**

1. 5 수 정렬하기 #2750 ★
2. 5 수 정렬하기 2 #2751
3. 5 수 정렬하기 3 #10989
4. 5 좌표 정렬하기 #11650 ★
5. 5 좌표 정렬하기 2 #11651
6. 5 나이순 정렬 #10814
7. 5 단어 정렬 #1181

✓ **문자열**

8. 5 모음의 개수 #10987
9. 5 알파벳 개수 #10808
10. 3 단어의 개수 #1152
11. 5 단어 공부 #1157

✓ **큐, 스택**

12. 4 큐 #10845 ★
13. 3 프린터 큐 #1966
14. 4 스택 #10828 ★
15. 4 괄호 #9012
16. 3 스택 수열 #1874

✓ **우선순위 큐**

17. 4 최소 힙 #1927
18. 4 최대 힙 #11279 ★
19. 4 절댓값 힙 #11286

✓ **???**

20. 5 집합 #11723
21. 4 덱 #10866
22. 4 포켓몬 마스터 이다솜 #1620