Introduction to STL and Data Structures

by Aulene De

This document is supposed to serve as a supplement to my original guide, but I didn't know how to make this, you know, not-feel-out-of-place.

This won't be comprehensive, (like all my works, LOL), so I encourage you to please take a look at CLRS or even the Competitive Programming Handbook for in-depth explanations.

Vectors (Dynamic Arrays)

Ever wanted to create an array and not have to declare the amount of space you need for it? INTROOOODUCCINGGGG - VECTORS!

What are Vectors? Basically, they're arrays, but you don't need to explicitly declare the size of a vector unlike a normal array. Their size is malleable throughout the execution of a program.

The main difference between a normal array and a vector is in insertion. Memory is not allocated for an element in a vector until you insert an element with the push_back() function. After that, you can the use the vector like an ordinary array.

Functions -

1. push_back(x) - inserts x element into the vector
2. size() - returns the number of elements in a vector
3. back() - returns the last element in a vector

```
1   vector <int> v;
2   v.push_back(4); // [4]
3   v.push_back(2); // [4, 2]
4   v.push_back(0); // [4, 2, 0]
5
6   cout << v[0] << endl; // 4
7   cout << v[1] << endl; // 2
8   cout << v[2] << endl; // 0
9
10  cout << v.size() << endl; // 3
11  cout << v.backK() << endl; // 0
12
13  for(i = 0; i < v.size(); i++)
14      cout << v[i] << " ";
```

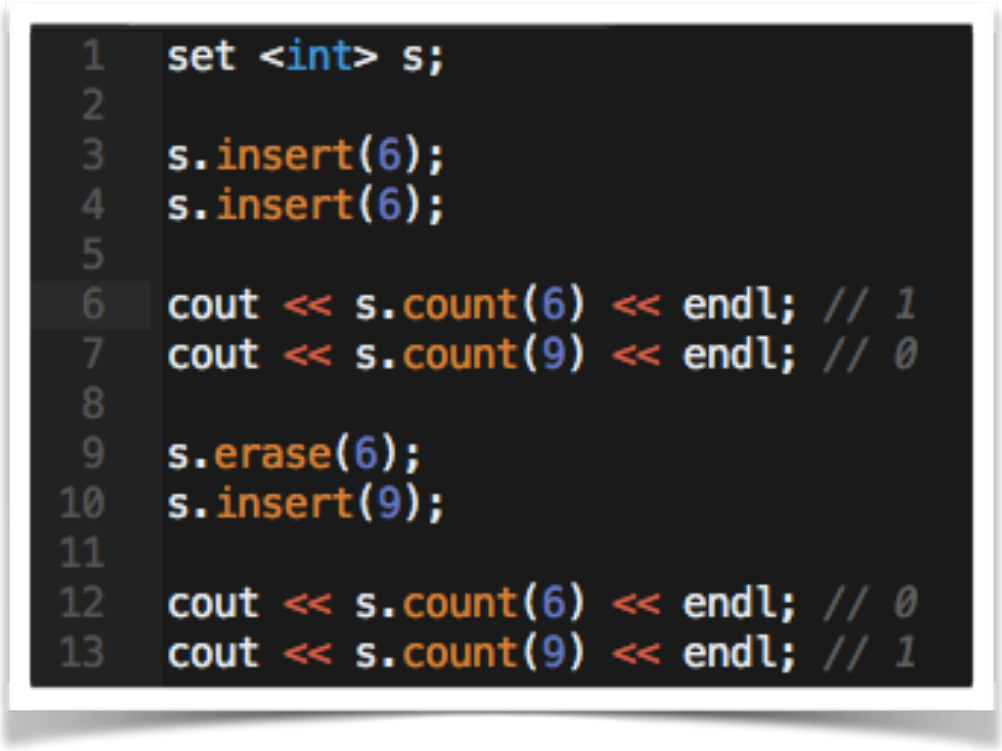
Sets and Multisets

Maps are structures that maintains a collection of elements.

Functions -

1. insert(x) - inserts x into the set.
2. count(x) - returns the number of elements with key x
3. erase(x) - removes all elements with key x

Basically, if you use a set, you can use it to find whether an element exists or not. :P



```
1   set <int> s;
2
3   s.insert(6);
4   s.insert(6);
5
6   cout << s.count(6) << endl; // 1
7   cout << s.count(9) << endl; // 0
8
9   s.erase(6);
10  s.insert(9);
11
12  cout << s.count(6) << endl; // 0
13  cout << s.count(9) << endl; // 1
```

Alternatively, you can use a multiset, which will give you the exact number of occurrences of an element.

```
multiset <int> s;

s.insert(6);
s.insert(6);

cout << s.count(6) << endl; // 2
cout << s.count(9) << endl; // 0

s.erase(6);
s.insert(9);

cout << s.count(6) << endl; // 0
cout << s.count(9) << endl; // 1

```

Problem
You are given a array of words. You are given a query as a word, and you have to answer the number of occurrences of that word in the array.

Solution
Pretty simple. We declare a string multiset, and answer queries using the count() function.

Maps

Maps are structures that maintain key-value pairs. What this means is that not only can you determine what type of data you want to store (the key), you can also determine how you want to store it (the value). Also, elements of a map can be accessed like a vector using the [] notation.

```
map <string, int> M;

M["a"] = 2;
M["b"] = 2;
M["c"] = 2;

cout << M["a"] + M["b"] << endl; // 4

```

Stack

A Stack operates on a Last-In-First-Out (LIFO) principle. It has two operations - add an element to the top, and remove an element from the top.

Functions -

1. push(x) - pushes x to the top of the stack
2. pop() - removes the first element of the stack
3. top() - returns the first element of the stack

Queue

A Queue operates on a First-In-First-Out (FIFO) principle. It has two operations - add an element to the top, and remove an element from the bottom.

Functions -

1. push(x) - pushes x to the back of the queue
2. pop() - removes the front element of the queue
3. front() - returns the front element of the queue

```
queue <int> q;

q.push(4);
q.push(2);
q.push(0);

cout << q.front() << endl; // 4
q.pop();
cout << q.front() << endl; // 2

```

```
stack <int> s;

s.push(4);
s.push(2);
s.push(0);

cout << s.top() << endl; // 0
s.pop();
cout << s.top() << endl; // 2

```

Priority Queues (Heaps)

Priority Queues are sets of elements that support insertion and deletion in $O(LogN)$ time. By default, Priority Queues are sorted in decreasing order (max-heap), with the largest element on the top. It is possible to modify them and use them as a min-heap as well.

Functions -

1. push(x) - pushes x into the priority queue
2. pop() - removes the element at the top of the priority queue (the greatest element in the case of a max-heap)
3. top() - returns the top element in the priority queue

```
priority_queue <int> pq;

pq.push(4);
pq.push(2);
pq.push(0);

cout << pq.top() << endl; // 4

pq.pop();

cout << pq.top() << endl; // 2

pq.pop();
pq.push(100);

cout << pq.top() << endl; // 100
pq.pop();
```

Fin.

Here's the thing with STL.

It's a lot easier to know it, and a lot harder to get used to actively using it in competitions. Try to incorporate using these structures actively in your codes and you'll find that what could be a 200-line ugly piece of code could be tuned down to a more efficient and sexier 50-line implementation.

Back to the guide.