EECS 281, Week 2: Wednesday

Templates

Function templates:

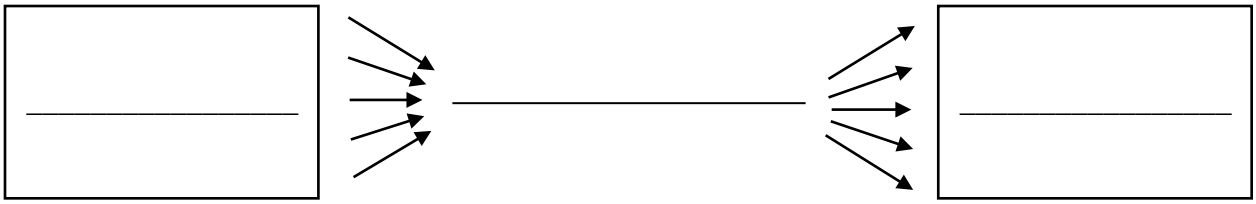
```
template <typename T>
void swapValues(T& a, T& b) {
    T temporaryA = a;
    a = b;
    b = temporaryA;
}
```

Class templates:

```
template <typename T>
class NamedStack {
    vector<T> items;
    string name;
public:
    NamedStack(string newName): name(newName) {};
    void push(T newItem) {
        items.push_back(newItem);
    }
    // ...
};
```

Difference between function templates and class templates:

Standard Template Library



_____ implementations for *A* algorithms and *C* containers.

Reasons to use STL:

EECS 281, Week 2: Wednesday

Iterators

Iterators are a pure abstraction!

Type	`it1 == it2` `it1 != it2`	`*it` `it->`	`++it`	`--it`	`it1 < it2` `it1 > it2`	`it + n` `it - n`	Providers
Forward							
Bidirectional							
Random-access							

Input iterator: _____

Output iterator: _____

Iterator Functions

```
advance(it, 5);
distance(it1, it2);
```

Iterator Adapters

Insert iterators

Stream iterators

Reverse iterators

Move iterators

EECS 281, Week 2: Wednesday

STL Containers

Type	Containers
	`vector, deque, list, forward list`
	`(multi)set, (multi)map`
	`unordered_(multi)set, unordered_(multi)map`
	`queue, stack, priority queue`

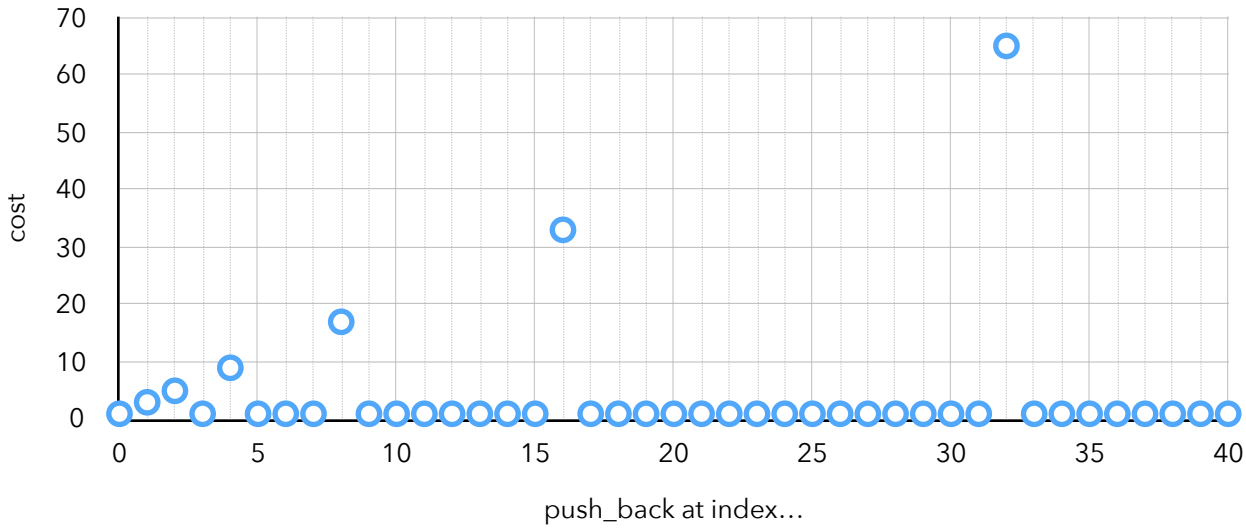
Vector

Deque

Array

vector::push_back()

push_back at i…	0	1	2	3	4	5	6	7	8
size									
capacity									
write cost									
resize cost									
total cost									
cumulative cost									
average cost									



Amortized Complexity

Definition: _____

_____

_____

EECS 281, Week 2: Wednesday

List

Forward List

Iterator Invalidation

Vector _____

Deque _____

List _____

Sequence Containers Summary

Container	insert/remove front	insert/remove middle	insert/remove back	search
Vector				
Deque				
Array				
List				
Forward List				

Associative Containers

Set

Multiset

Map

Multipap

Unordered Associative Containers

Unordered set

Unordered multiset

Unordered map

Unordered multipap

Find Sum

Given an integer `sum` and a sorted array `numbers` of `N` distinct integers, implement a function to find if there exist indices `i` and `j` such that `numbers[i] + numbers[j] == x`.

```
bool findSum(const vector<int> &numbers, int sum) {
    for (int i                                          ) {
        for (int j                                            ) {
            if (                                        ) {

            }
        }
    }
    return
}
```

Implement a better, more efficient version of the algorithm.

```
bool betterFindSum(const vector<int> &numbers, int sum) {









}
```

Algorithms

Functions that mostly involve loops.

Here are some examples:

`min_element`
`sort`
`partial_sort`
`reverse`
`copy`
`find_if`
`for_each`

Anagrams

Implement `areAnagrams`, a function that checks if `s1` and `s2` are anagrams of each other.

For example, the strings `"cinema"` and `"iceman"` are anagrams. Feel free to use (other) STL containers and algorithms.

```
bool areAnagrams(const string& s1, const string& s2) {









}
```

When finished, take a picture of your code and email it to Maxim.