Please do not distribute or host these slides without prior permission.
STL Data Structures
What data structure(s) have we learned about so far in this class?
What were the tradeoffs associated with the data structure?
Writing them down in a table is a nice exercise to do in your spare time to really understand them!
Lecture
Figure 1: Marissa Mayer is a Stanford Graduate (Masters in CS) and was the 20th employee at Google. She was Vice President of Search while at Google, and was most recently the CEO of Yahoo. During her time at Google, she also taught an introductory programming class—similar to the one you are in now!
In the previous lecture, we learned about files.
Before that, we learned about functions, arrays, and strings.
As you may have noticed, Computer Scientists like coming up with abstractions to make life easier.

In this lecture, we are going to learn about more!
How do we resize an array?

- Previously, we have created arrays in the following way.
- `int myArray[10];`
One possible strategy, is to create another array, say `arrayCopy`.

```c
int arrayCopy[20];
```

We would then have to set the first ten elements of our previous array, into the first ten indicies of the `arrayCopy`.

This is actually an okay strategy.
But now I changed my mind

- Let's now say that I actually do not need that much memory.
- I have changed my mind yet AGAIN!
- Well, that is okay, programmers are allowed to do that.
- Should I go through the trouble of downsizing my array?
- What if I change my mind again?
std::vector
A `std::vector` is a data structure that allows us to add and remove one piece of data at a time.

- It stores one kind of data type just like an array.
- We will think of a `std::vector` as a templated array, it can hold any datatype we tell it to.
- It is templated, so the syntax to a template function will be similar to us.
Vector - A first example

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

int main() {
    // Create a new vector
    // Note that we do not need to specify the size
    // of the vector.
    // We do need to specify the type though.
    std::vector<int> myVectorOfInt;
    // We push back (i.e. append to our structure)
    // Our first integer.
    myVectorOfInt.push_back(7);
    std::cout << "Our first element is: " << myVectorOfInt[0] << "\n";
    return 0;
}
```

Listing 1: Vector is a dynamically sizeable array
It is worth noting, we can do a lot with vectors!

<table>
<thead>
<tr>
<th>Element access:</th>
<th>Modifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator[]</code></td>
<td>Access element (public member function)</td>
</tr>
<tr>
<td><code>at</code></td>
<td>Access element (public member function)</td>
</tr>
<tr>
<td><code>front</code></td>
<td>Access first element (public member function)</td>
</tr>
<tr>
<td><code>back</code></td>
<td>Access last element (public member function)</td>
</tr>
<tr>
<td><code>data</code></td>
<td>Access data (public member function)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifiers:</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>assign</code></td>
<td>Assign vector content (public member function)</td>
</tr>
<tr>
<td><code>push_back</code></td>
<td>Add element at the end (public member function)</td>
</tr>
<tr>
<td><code>pop_back</code></td>
<td>Delete last element (public member function)</td>
</tr>
<tr>
<td><code>insert</code></td>
<td>Insert elements (public member function)</td>
</tr>
<tr>
<td><code>erase</code></td>
<td>Erase elements (public member function)</td>
</tr>
<tr>
<td><code>swap</code></td>
<td>Swap content (public member function)</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>Clear content (public member function)</td>
</tr>
<tr>
<td><code>emplace</code></td>
<td>Construct and insert element (public member function)</td>
</tr>
<tr>
<td><code>emplace_back</code></td>
<td>Construct and insert element at the end (public member function)</td>
</tr>
</tbody>
</table>

Figure 2: A subset of member functions we can use with vectors
```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> myVectorOfInt;
    // Add some elements
    myVectorOfInt.push_back(1);
    myVectorOfInt.push_back(1);
    myVectorOfInt.push_back(2);
    myVectorOfInt.push_back(3);
    myVectorOfInt.push_back(5);
    myVectorOfInt.push_back(8);
    // Using size() to tell us how big our vector is
    for (int i = 0; i < myVectorOfInt.size(); ++i) {
        std::cout << myVectorOfInt[i] << " \n";
    }

    return 0;
}
```

Listing 2: Vector iteration
Vector - Remove an item from vector

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

int main() {
    std::vector<int> myVectorOfInt;
    myVectorOfInt.push_back(1);
    myVectorOfInt.push_back(1);
    myVectorOfInt.push_back(2);
    myVectorOfInt.push_back(3);
    myVectorOfInt.push_back(5);
    myVectorOfInt.push_back(8);
    // Delete the last element in the vector.
    myVectorOfInt.pop_back();
    // Delete the last element again
    myVectorOfInt.pop_back();

    for (int i = 0; i < myVectorOfInt.size(); ++i) {
        std::cout << myVectorOfInt[i] << "\n";
    }
    return 0;
}
```

Mike Shah (Tufts University)
Behind the scenes, the vector is doing some book keeping.

A vector internally keeps track of how many items we have 'pushed' into it.

It also keeps track of where that data is in memory for us.

Because again, arrays are contiguous blocks of memory they cannot be resized on demand. The vector is an abstraction on top of an array that allows us to do this at runtime.
On your own look at the member functions and try some of the examples out.

- It keeps track of how many items we have ‘pushed’ into it.
- It also keeps track of where that data is in memory for us.
- Because again, arrays are contiguous blocks of memory, with some pieces of data.
Stack
Stack

- A stack is a data structure that reflects exactly what it sounds like.
- A stack of your (roommates/significant other/friends/neighbors) dirty dishes
- Let's investigate some more.

Figure 3: Stacking dishes is a direct analogy to the stack data structure
Stack of dirty dishes

- You and your friends have a dinner party
Stack of dirty dishes

- You and your friends have a dinner party
- The first person to finish their dinner puts their plate by the sink (Larry).

Figure 4: push() Larry on the stack
You and your friends have a dinner party.
The first person to finish their dinner puts their plate by the sink (Larry).
The next person puts their dish right on top (Curly).

Figure 5: push() Curly on the stack
You and your friends have a dinner party.

The first person to finish their dinner puts their plate by the sink (Larry).

The next person puts their dish right on top (Curly).

Then another person puts their dish on top (Moe).

Figure 6: push() Moe on the stack
When you wash the dishes, you start cleaning by washing the dish on the top of the stack.

Figure 7: The full stack
When you wash the dishes, you start cleaning by washing the dish on the top of the stack.

The last person to put their dish on the stack was Moe.

Figure 8: pop() Moe i.e. wash Moe’s dish
When you wash the dishes, you start cleaning by washing the dish on the top of the stack.

Now, the last person to put their dish on the stack was Curly.

Figure 9: pop() Curly i.e. wash Curly’s dish
When you wash the dishes, you start cleaning by washing the dish on the top of the stack. Finally, you can wash Larry’s dish, as he is last on the stack.

(We popped Larry, and washed his dish)
No more dishes to wash!
Now imagine a stack with 1000 dishes. You cannot simply pull a dish from the middle of a stack, it would fall over!

This is the same in C++, you must wash dishes in this order.

This is known as a LIFO data structure. Last-in, First-out (LIFO).

The last dish in, is the first one to get washed out.

Time for some code!
Stack - First example

```cpp
#include <iostream>
#include <string>
// New library
#include <stack>

int main()
{
    // Create a stack that holds strings.
    // Again, a stack holds one type.
    std::stack<std::string> myStackOfDishes;

    myStackOfDishes.push("Larry");
    myStackOfDishes.push("Curly");
    myStackOfDishes.push("Moe");
    std::cout << "stack size:" << myStackOfDishes.size() << "\n";
    // Pop removes whatever is on top of the stack.
    // No fancy index, or need to specify a specific element.
    myStackOfDishes.pop();
    myStackOfDishes.pop();
    myStackOfDishes.pop();
    // Stack size after removing all items.
    std::cout << "stack size:" << myStackOfDishes.size() << "\n";
    return 0;
}
```

**Listing 4:** Adding items to a stack and then removing them
Stacks have several member functions as well.

Figure 10: A subset of member functions we can use with Stacks

<table>
<thead>
<tr>
<th>Member function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(constructor)</strong></td>
<td>Construct stack (public member function)</td>
</tr>
<tr>
<td><code>empty</code></td>
<td>Test whether container is empty (public member function)</td>
</tr>
<tr>
<td><code>size</code></td>
<td>Return size (public member function)</td>
</tr>
<tr>
<td><code>top</code></td>
<td>Access next element (public member function)</td>
</tr>
<tr>
<td><code>push</code></td>
<td>Insert element (public member function)</td>
</tr>
<tr>
<td><code>emplace</code></td>
<td>Construct and insert element (public member function)</td>
</tr>
<tr>
<td><code>pop</code></td>
<td>Remove top element (public member function)</td>
</tr>
<tr>
<td><code>swap</code></td>
<td>Swap contents (public member function)</td>
</tr>
</tbody>
</table>
Stack - top

- Stack appears relatively simplistic, and that is for reasons of speed and ease of use.
- If we actually want to look at the items we pop off, we first look at the top() of the stack, and then pop it.
Listing 5: This example shows how to iterate through an entire stack.
STL
Both the stack and the vector are part of the C++ standard template library.

These are data structures provided to us to make our lives easier. They are well tested and thought out.

There exist many other data structures in the STL (queues, maps, and more). Explore!

For this class, using the STL is expected and encouraged!
Activity Discussion
Review of what we learned

- (At least) Two students
- Tell me each 1 thing you learned or found interesting in lecture.
5-10 minute break
To the lab!

No lab today–there will be a full lab on Wednesday.

You should have gotten an e-mail and hopefully setup an account at https://www.eecs.tufts.edu/~accounts prior to today. If not–no worries, we'll take care of it during lab!

1