Comp 11 Lectures

Mike Shah

Tufts University

June 18, 2017

STL Data Structures

Comp 11 - Pre-Class warm up

- 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

Marissa Mayer

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!

More Abstraction

- 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];

Make a bigger array

- One possible strategy, is to create another array, say arrayCopy.
- 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

- Lets 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

Vector - a resizable array

- 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

```
1  #include <iostream>
2  // include our new vector library
3  #include <vector>
4
5  int main(){
6    // Create a new vector
7    // Note that we do not need to specify the size
8    // of the vector.
9    // We do need to specify the type though.
10   std::vector<int> myVectorOfInt;
11   // We push back (i.e. append to our structure)
12   // Our first integer.
13   myVectorOfInt.push_back(7);
14   std::cout << "Our first element is: " << myVectorOfInt[0]
       << "\n";
15
16   return 0;
17 }
```

Listing 1: Vector is a dynamically sizeable array

Vector - Member Functions

It is worth noting, we can do a lot with vectors!

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

Figure 2: A subset of member functions we can use with vectors

Vector - Iterating through a vector

```
1  #include <iostream>
2  #include <vector>
3
4  int main(){
5    std::vector<int> myVectorOfInt;
6    // Add some elements
7    myVectorOfInt.push_back(1);
8    myVectorOfInt.push_back(1);
9    myVectorOfInt.push_back(2);
10   myVectorOfInt.push_back(3);
11   myVectorOfInt.push_back(5);
12   myVectorOfInt.push_back(8);
13   // Using size() to tell us how big our vector is
14   for(int i=0; i < myVectorOfInt.size(); ++i){
15     std::cout << myVectorOfInt[i] << "\n";
16   }
17
18   return 0;
19 }
```

Listing 2: Vector iteration

Vector - Remove an item from vector

```
#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;
```

Vector - How does it work?

- 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.

Vector - More to experiment with

- 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 struture that reflects exactly what it sounds like.
- A stack of your (roommates/signficant other/friends/neighbors) dirty dishes
- Lets 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

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

Figure 5: push() Curly on the stack

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).
- 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

Stack of dirty dishes - cleanup

- When you wash the dishes, you start cleaning by washing the dish on the top of the stack.

Figure 7: The full stack

Stack of dirty dishes - cleanup

- 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

Stack of dirty dishes - cleanup

- 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

Stack of dirty dishes - cleanup

- 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!

Stack of dirty dishes - LIFO

- 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

```
#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 fter removing all items.
  std::cout << "stack size:" << myStackOfDishes.size() << "\n";
  return 0;
}
```

Listing 4: Adding items to a stack and then removing them

Stack - Member Functions

Stacks have several member functions as well.

Member functions

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

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

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.

Stack - top and empty example

```
1  #include <iostream>
2  #include <string>
3  // New library
4  #include <stack>
5
6  int main(){
7    std::stack<std::string> myStackOfDishes;
8
9    myStackOfDishes.push("Larry");
10   myStackOfDishes.push("Curly");
11   myStackOfDishes.push("Moe");
12   // Note the ! operator.
13   // This says, while our stack is not empty.
14   while(!myStackOfDishes.empty()){
15   std::cout << "Popping off:" << myStackOfDishes.top() << '\n';
16     myStackOfDishes.pop();
17   }
18
19   return 0;
20 }
```

Listing 5: This example shows how to iterate through an entire stack.

STL

STL - Standard Template Library

- 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!

In-Class Activity

http://www.mshah.io/comp/11/activities/activity7/activity.pdf

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.
[1]

[1] 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!