Sequence Containers

...

Original from : Mike Precup (mprecup@cs.stanford.edu)
ENJMIN Edition 2016

Structs

- A struct is an easy way to bundle multiple variables together

Structs

```
struct point {
    int x;
    int y;
};

point p;
p.x = 4;
p.y = 3;
```

Review: Sequence Containers

- A container class allows you to store any number of things
- A sequence container is a container whose elements can be accessed sequentially.
- Sequence containers include vectors, lists, and more!
- Container adaptors include stacks, queues, and priority queues
- http://www.cplusplus.com/reference/stl/

Container #1: Stack

First, let's talk about how

to use the STL stack.

STL <stack>:

What you want to do	STL stack<int>
Create a stack	`stack<int> x;`
Get the size of a stack	`int size = x.size();`
Check if a stack is empty	`if (x.empty()) ...`
Push a value on the stack	`x.push(42);`
Peek at the top element without popping it	`int top = x.top();`
Pop off the top element and ignore its value	`x.pop();`

STL <stack>:

What you want to do	STL stack<int>
Clear the stack	`while(!x.empty())` `  x.pop();`

Convert the stack to a string	`string s;`
	`while(!x.empty() {`
	`  s += x.top();`
	`  s += " ";`
	`  x.pop();`
	`}`
	`// doesn't work for int`

Pop and save the value	`int top = x.top();` `x.pop();`

STL <stack>: Usage

Let's look at a quick demo in `STLStack`

“Thus, the standard library will serve as

both a tool and as a teacher”

- Bjarne Stroustrup

STL `<stack>`: Why ?

Why is there no `.clear()` function for stacks?

STL <stack>: Why ?

Why is there no `.clear()` function for stacks?

- Conceptually, clearing isn't part of the interface to a stack
- It's very easy to write your own clear function:

```
// stack<int> s = ...;
while (!s.empty()) {
  s.pop();
}
```

STL `<stack>`: Why ?

Why doesn't pop return the value it removed?

STL <stack>: Why ?

Why doesn't pop return the value it removed?

- The caller might not need the value, in which case returning the value would be wasteful.
- It's easy to write code which pops and saves the value.

```
// stack<int> s = ...;

int value = s.top();

s.pop();
```

STL `<stack>`: Why ?

Why isn't there a `toString` function?

STL `<stack>`: Why ?

Why isn't there a `toString` function?

- Implementing toString would require that the type stored in the stack could be converted to a string
    - For example, you can convert a `stack<int>` to a `string` because you can convert an `int` to a `string`.
- It's tough to say what the "proper" way to convert a stack to a string is

Container #2: Vector

First, let's talk about how vectors are represented in the STL.

STL <vector>:

What you want to do	STL vector<int>
Create an empty vector	`vector<int> v;`
Create a vector with n copies of zero	`vector<int> v(n);`
Create a vector with n copies of a value k	`vector<int> v(n, k);`
Add a value k to the end of the vector	`v.push_back(k);`
Clear a vector	`v.clear();`
Get the element at index i (verify that i is in bounds)	`int k = v.at(i);`
Check if the vector is empty	`if (v.empty()) ...`
Replace the element at index i (verify that i is in bounds)	`v.at(i) = k;`

STL <vector>:

Get the element at index i without bounds checking	`int a = x[i];`
Change the element at index i without bounds checking	`x[i] = v;`
Apply a function to each element in x	`// We'll talk about this in another lecture...`
Concatenate vectors v1 and v2	`// We'll talk about this in another lecture...`
Add an element to the beginning of a vector	`// Impossible! (or at least slow)`

STL <vector>: Usage

Let's look at a quick demo in `STLVector`

STL <vector>: Why ?

Why doesn't vector have bounds checking?

STL <vector>: Why ?

Why doesn't vector have bounds checking?

- If you write your program correctly, bounds checking will do nothing but make your code run slower

STL `<vector>`: Why ?

Why is there no `push_front` method?

STL `<vector>`: Why ?

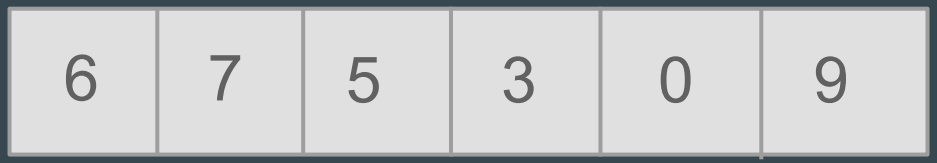
Why is there no `push_front` method?

- This is a bit more complicated

The Mystery of `push_front`

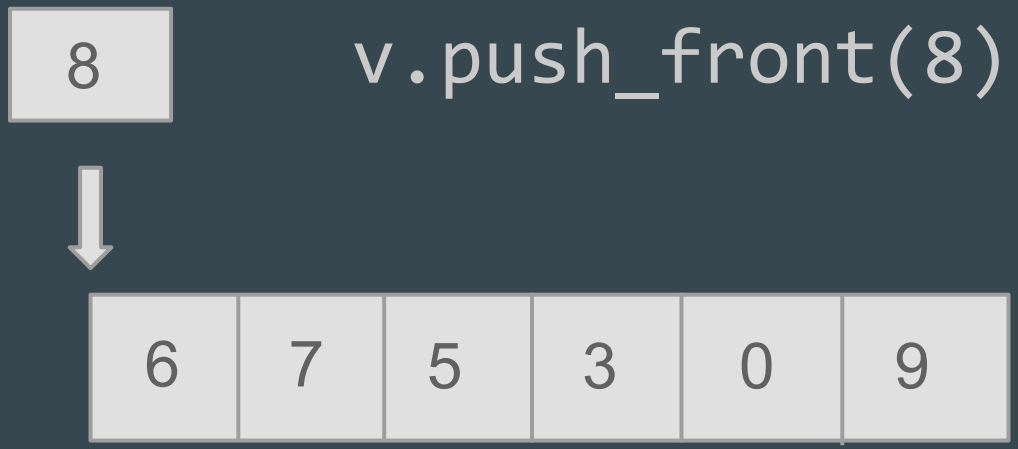
Pushing an element to the front of the vector requires shifting all other elements in the vector down by one, which can be **very** slow

To demonstrate this, let's say we had this nice little vector:



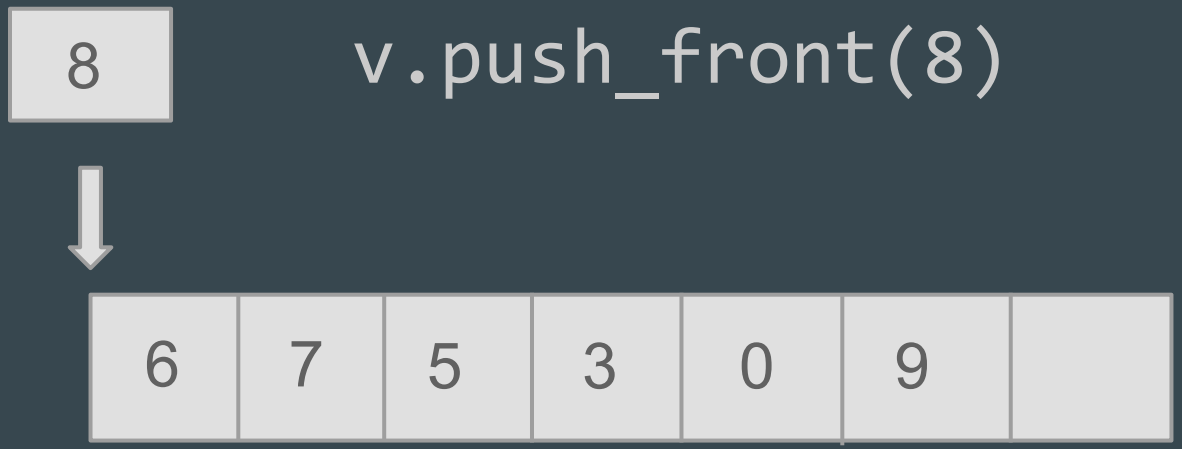
The Mystery of `push_front`

Now, let's say that `push_front` existed, and that you wanted to insert an 8 at the beginning of this vector.



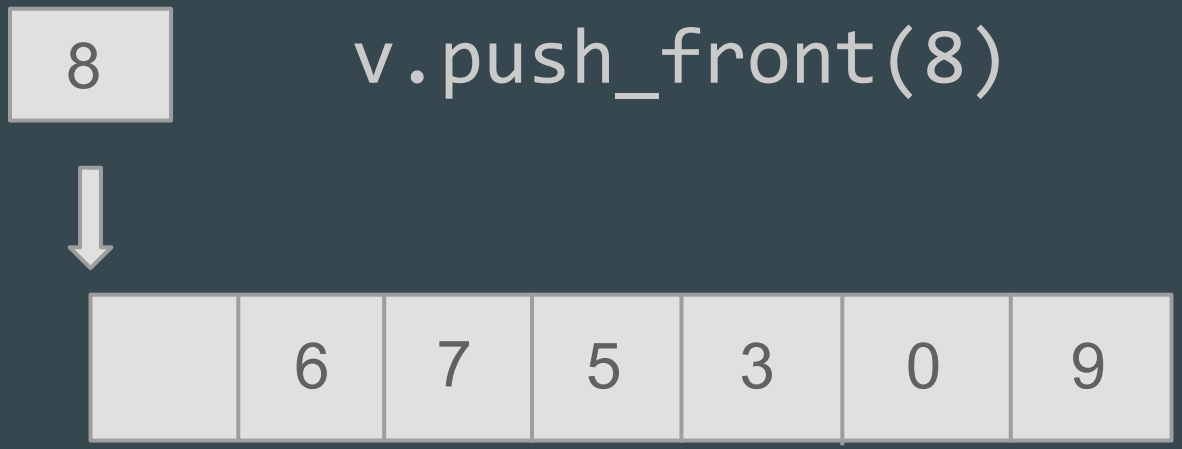
The Mystery of `push_front`

First, we may have to expand the capacity of the vector



The Mystery of `push_front`

Then, we'll need to shift every single element down one position



The Mystery of `push_front`

Finally, we can actually insert the element we wanted to insert.

```
v.push_front(8)
```

Just how bad is push_front?

```
// Adding to the back
for (int i = 0; i < N; i++)
    v.push_back(i);


// Or: Adding to the front
for (int i = 0; i < N; i++)
    v.insert(v.begin(), i);


// How big can the difference be?
```

Just how bad is push_front?

	`push_front`	`push_back`
N = 1000	0.01	0
N = 10000	0.89	0.01
N = 100000	117.98	0.04
N = 1000000	Hours	0.31
N = 10000000	Months	3.16

You can see the difference between an $O(n^2)$ algorithm and an O(n) algorithm!

STL <deque>: What's a deque?

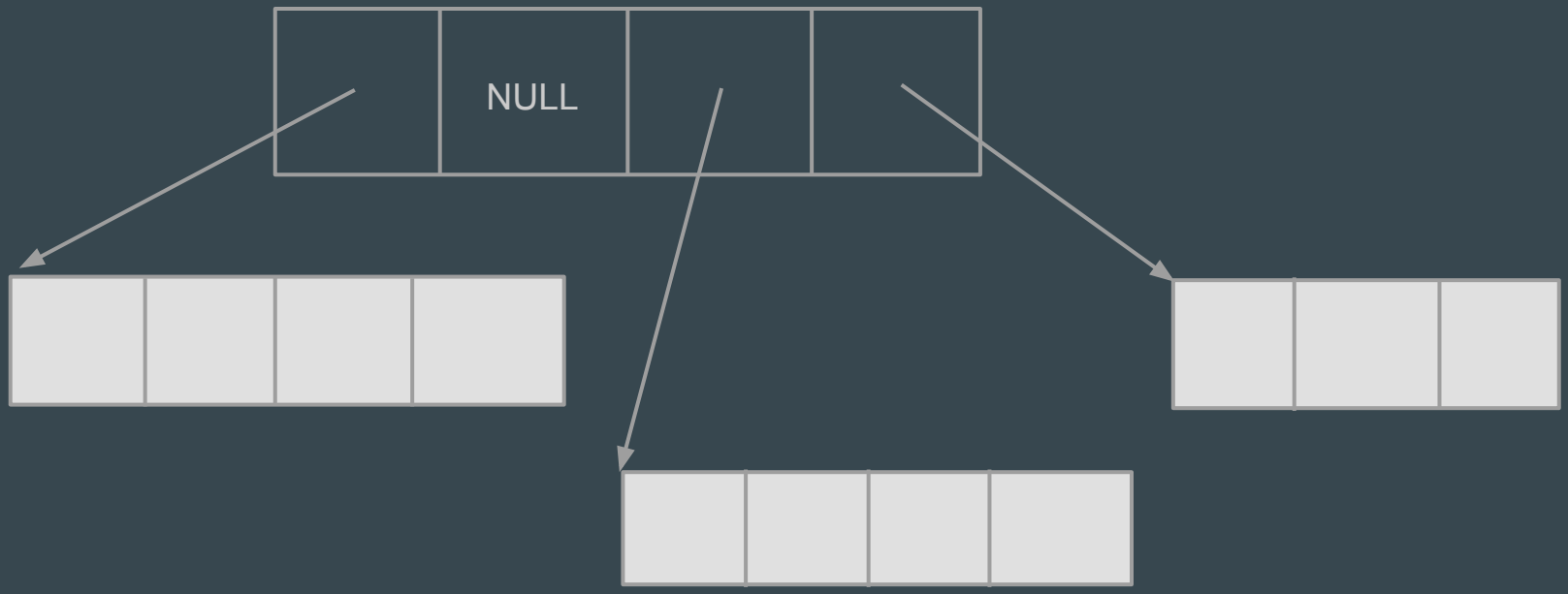
- A deque (pronounced "deck") is a **d**ouble **e**nded **que**ue
- Unlike a vector, it's possible (and fast) to `push_front`
- The implementation of a deque isn't as straightforward as a vector though

STL <deque>: Usage

Let's look at a quick demo in `STLDeque`

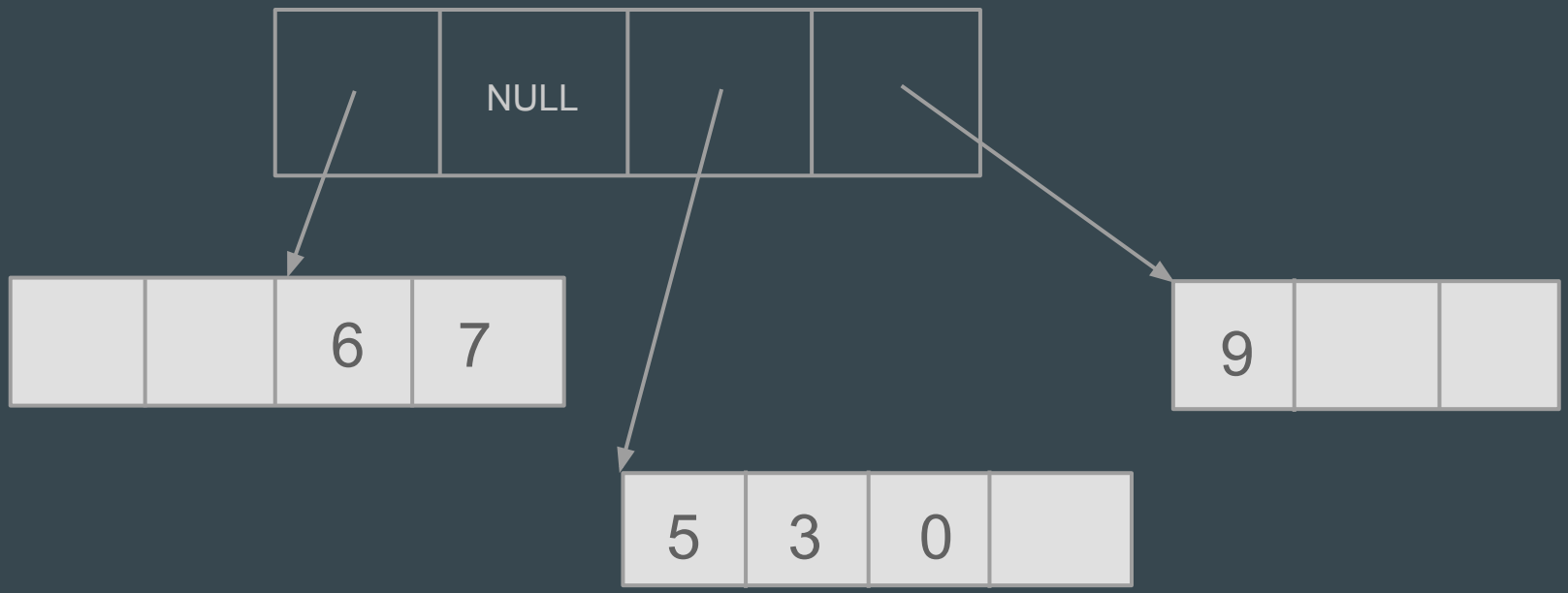
STL <deque>: Implementation

There's no single specification for representing a deque, but it might be laid out something like this



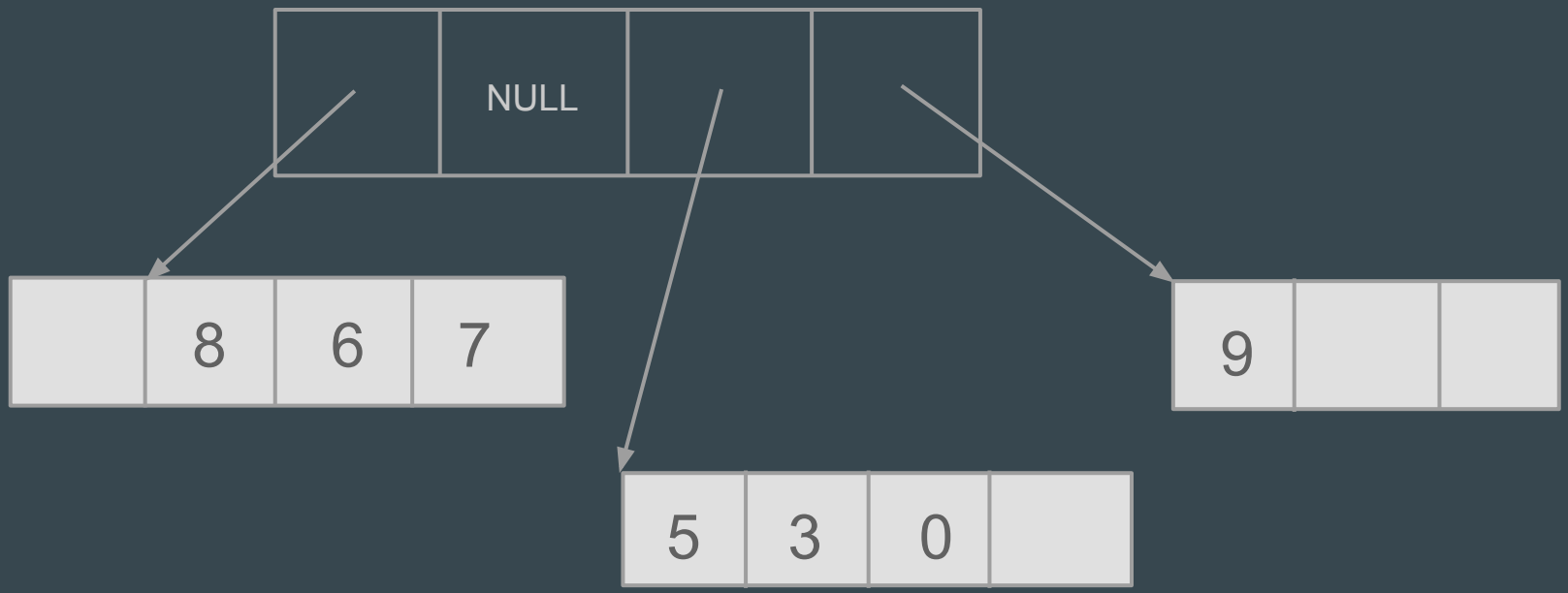
STL <deque>: Implementation

You could support efficient insertion by keeping some reserved space in front of the vector representing the first elements of the deque



STL <deque>: Implementation

You could support efficient insertion by keeping some reserved space in front of the vector representing the first elements of the deque



STL `<deque>`: Performance

- We can now use the `push_front` function, and it will run much faster than if we had used a vector.
- Let's see how this looks in real world performance numbers

push_front: vector and deque

```
// Vector test code

vector<int> v;

// Insert at the start of the vector

for (int i = 0; i < N; i++)

    v.insert(v.begin(), i);

// Clear by using pop_front (erase)

for (int i = 0; i < N; i++)

    v.erase(v.begin());
```

push_front: vector and deque

```
// Deque test code

deque<int> d;

// Insert elements using push_front

for (int i = 0; i < N; i++)

    d.push_front(i);

// Clear by using pop_front

for (int i = 0; i < N; i++)

    d.pop_front();
```

push_front: vector and deque

	<vector>	<deque>
N = 1000	0.02	0
N = 10000	2.12	0.01
N = 100000	264.9	0.04
N = 1000000	Hours	0.44
N = 10000000	Months	5.54

Why use a vector?

If a deque can do everything a vector can plus add to the beginning, why not always user deques?

Why use a vector?

If a deque can do everything a vector can plus add to the beginning, why not always user deques?

- For other common operations like access and adding to the end, a vector outperforms a deque

Element Access: vector and deque

```
vector<int> v(N);

deque<int> d(N);


for (int i = 0; i < N; i++)

    v[i] = i;


for (int i = 0; i < N; i++)

    d[i] = i;
```

Access: vector and deque

	<vector>	<deque>
N = 1000	0.02	0.14
N = 10000	0.28	1.32
N = 100000	3.02	13.22
N = 1000000	30.84	133.30

push_back: vector and deque

```
// Vector test code

vector<int> v;

// Insert elements using push_back

for (int i = 0; i < N; i++)

    v.push_back(i);

// Clear by using pop_back

for (int i = 0; i < N; i++)

    v.pop_back();
```

push_back: vector and deque

```
// Deque test code

deque<int> d;

// Insert elements using push_back

for (int i = 0; i < N; i++)

    d.push_back(i);

// Clear by using pop_back

for (int i = 0; i < N; i++)

    d.pop_back();
```

push_back: vector and deque

	<vector>	<deque>
N = 1000	0.02	0.02
N = 10000	0.20	0.20
N = 100000	1.98	1.92
N = 1000000	19.9	20.78

Showing Intent

- Why use a vector when you could use a deque (aside from performance)?
- Why use a stack when you could use a vector?