Vectors

CS 1044

Array Review

- Remember, arrays are collections of values
- But, some limitations:
  - What if you want the number of elements to depend on information not known at compile-time?
  - What if you want to **change the number** of elements while the program is running?
  - What if you want to **insert** or **remove** an element?

Aggregate Operations

- Here are some operations I might want to do with an array. Can I?

```
int x[100];
int y[100];
```

- `x = y;` `// assign the elements in y to x`
- `x == y;` `// check if x and y have the same elements`
- `return x;` `// return array from a function`

No to all three!

Vectors

```
#include <vector>
```

- To get around these limitations, C++ provides the `vector` data type
- `vector`s are collections that can **grow** and **shrink**
- Can also easily **insert and remove** elements from the beginning, middle, or end
- Think of arrays as fixed buckets while vectors are more flexible lists

The `vector` Data Type

- As with arrays, you give the element type when you declare an `vector` variable, but the syntax is different

```
vector<int> v;
```

Element type

- The <> denotes a **template** – a data type that depends on other data types
- Writing `vector` by itself without an element type is meaningless, and a syntax error

Using a Vector

- Conceptually, you can visualize an `vector` as a linear collection of slots, just like arrays
- Thanks to the flexibility of C++, you can use the same `array[i]` notation to get/set elements
- We can also retrieve the number of elements from the vector itself – no need to pass it around separately

Declaring a Vector

- We can give a vector an initial size:

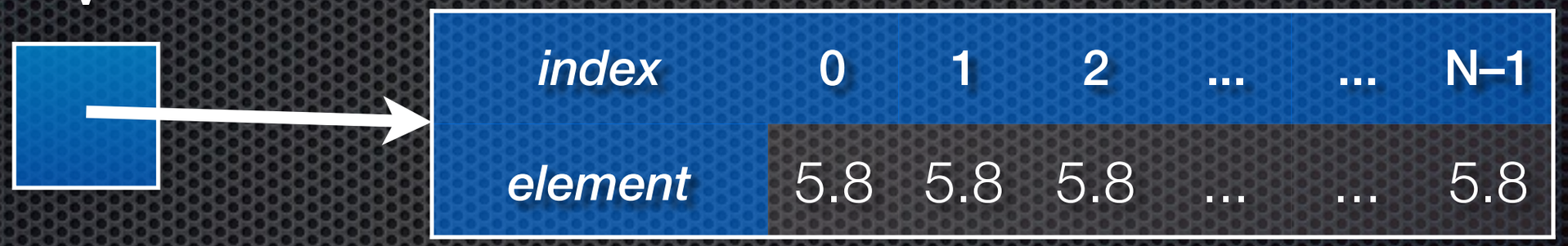
```
vector<int> v(N);
```

- This creates a vector with N elements, but the size can change later as well
- Unlike arrays, vector elements **are given default values** (0 for numbers, empty string for strings...)
- Can immediately access elements `v[0]` through `v[N - 1]`

Initializing Elements of a Vector

- After the size, we can give an optional default value for every element:

```
vector<double> v(N, 5.8);
```

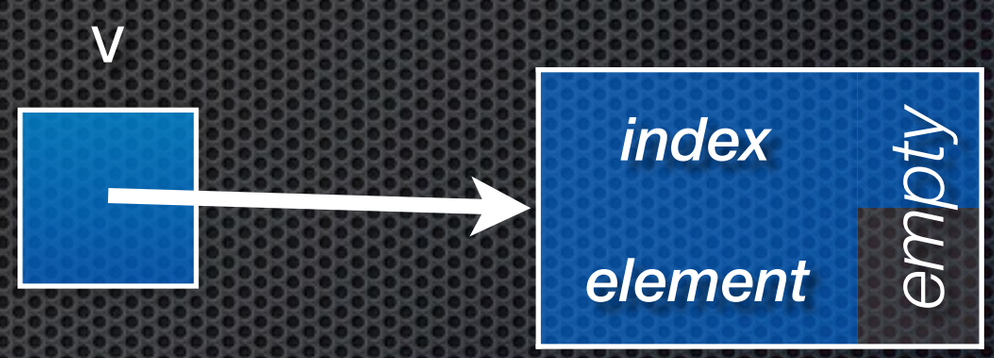


- Considerably more convenient than using a `for`-loop to initialize every element
- But, only useful if you want **every element** to have the **same value** initially

Empty Vectors

- Declaring a vector without any parameters makes it empty by default

```
vector<int> v;
```

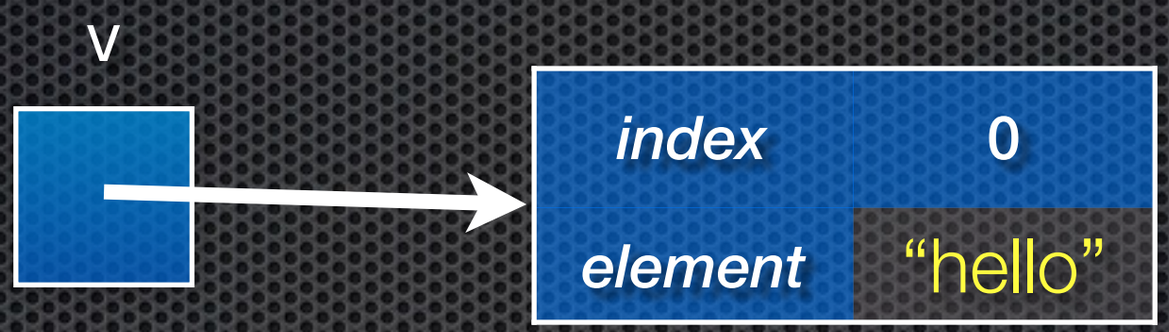


- Length is zero, so you can't access any elements, not even `v[0]`
- So how do we grow the vector?

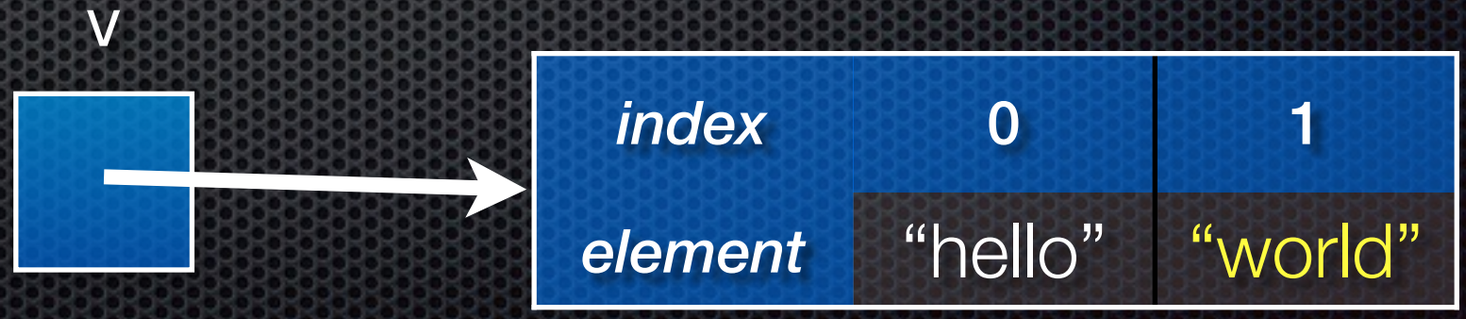
Adding to a Vector

```
v.push_back("hello");
```

`push_back` always adds to the end of the vector



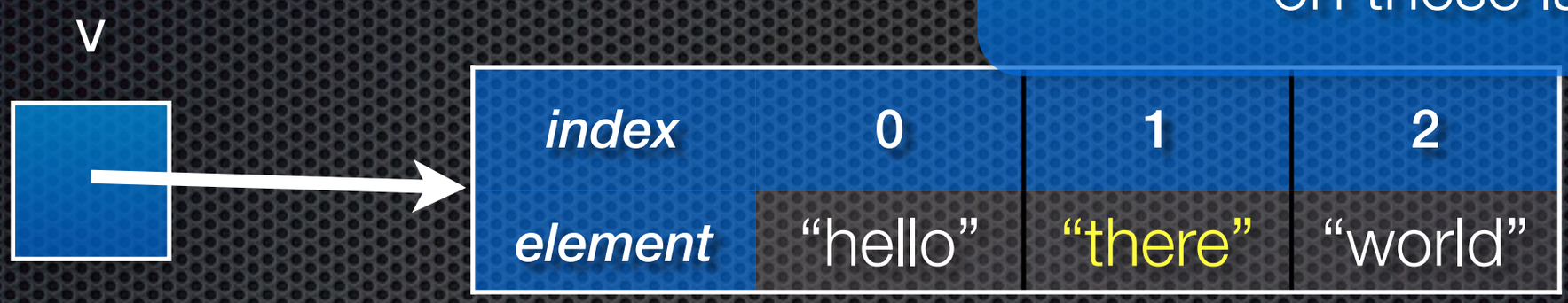

```
v.push_back("world");
```



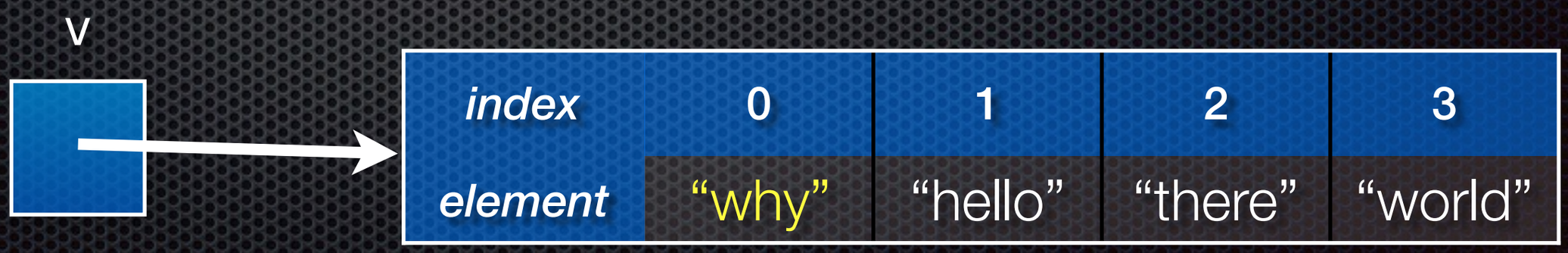
Inserting into the Middle

```
v.insert(v.begin() + 1, "there");
```

`v.begin()` is an **iterator** – more on these later



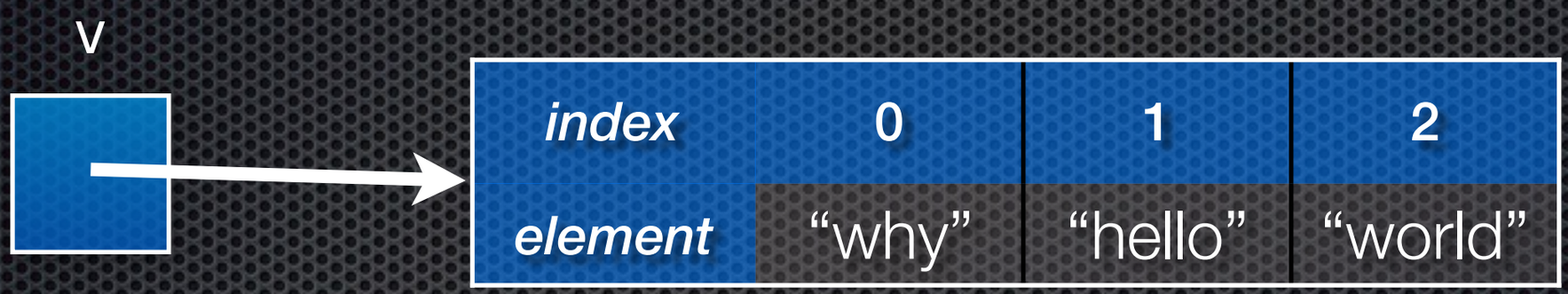

```
v.insert(v.begin(), "why");
```



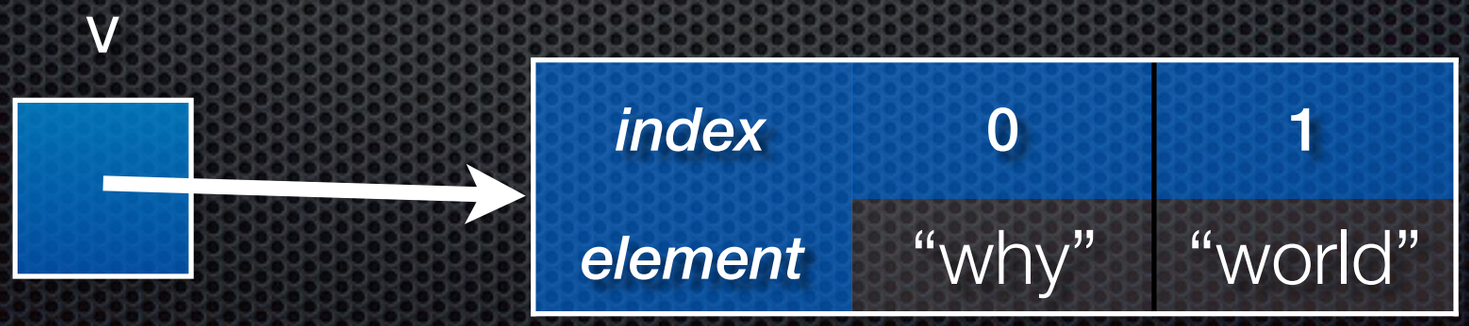
Erasing Elements

```
v.erase(v.begin() + 2);
```

Removes the element at the given position



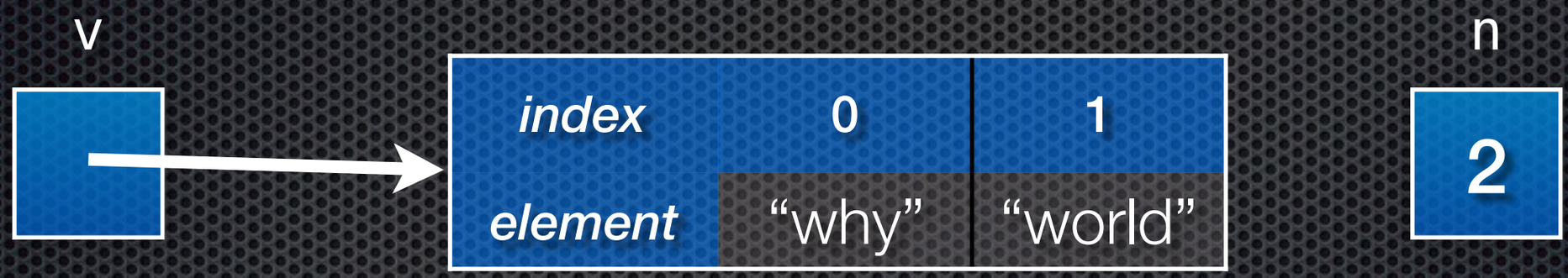

```
v.erase(v.begin() + 1);
```



Reading from a Vector

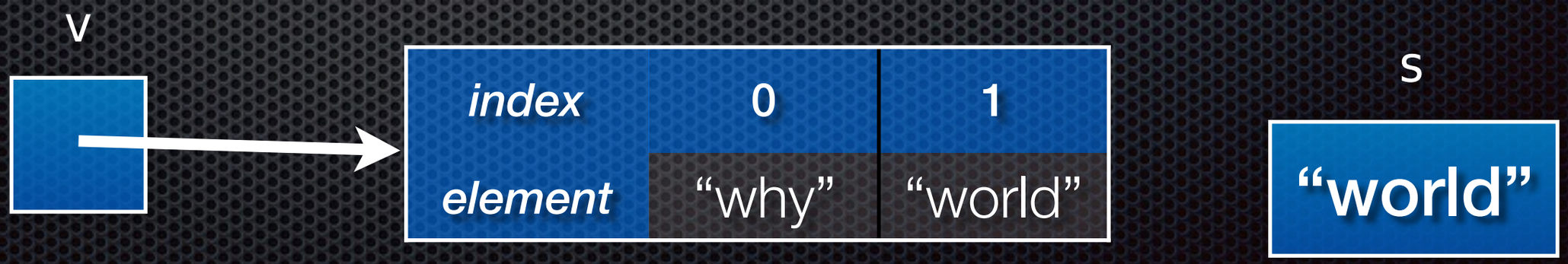
```
int n = v.size();
```

Gets the number of elements




```
string s = v[1];
```

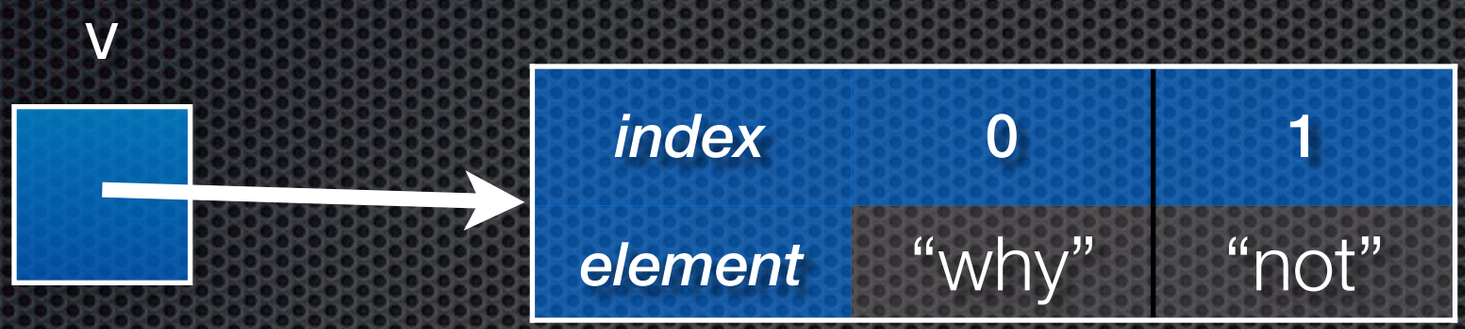
Gets the element at the given index



… and a Couple More

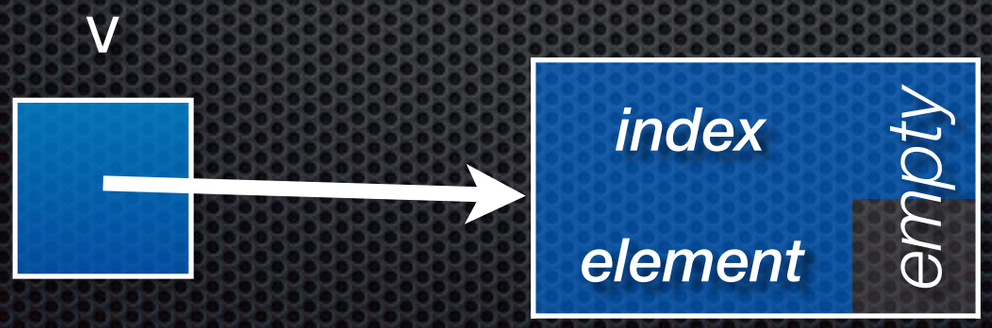
```
v[1] = "not";
```

Replaces the element at the given index




```
v.clear();
```

Removes everything from the vector



Aggregate Operations

- Look back at the aggregate operations that failed with arrays. Can we do them with vectors?

```
vector<int> x(100);
vector<int> y(100);
```

- `x = y;` `// assign the elements in y to x`
- `x == y;` `// check if x and y have the same elements`
- `return x;` `// return vector from a function`

Yes to all three!

When to Use Arrays or Vectors

- You should pretty much always use `vectors`
- But you need to be aware of arrays because you might have to work on code that uses them
- On the other hand, two-dimensional (and higher) `vectors` would be a pain – avoid them if you can