

Collections, Part One

Outline for Today

- ***Parameter Passing in C++***
 - On xeroxes and master copies.
- ***Container Types***
 - Holding lots of pieces of data.
- ***The Vector type***
 - Storing sequences.
- ***Recursion on Vectors***
 - More practice with sequences.

Parameter Passing in C++

Make a Prediction!

- Look over this piece of C++ code:

```
void becomeWealthy(int netWorth) {  
    netWorth = 10000000000;  
}  
  
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl; // <-- Here  
    return 0;  
}
```

- What do you think will get printed at the indicated point?

How it Works

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

How it Works

```
int main() {  
    int value = 137;  
    becomewealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

How it Works

```
int main() {  
    int value = 137;  
    becomewealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

137

value

How it Works

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

137

value

How it Works

```
int main() {
```

```
    int netWorth = 137;
```

137

```
    void becomeWealthy(int netWorth) {  
        netWorth = 10000000000;  
    }
```

137

netWorth

```
}
```

How it Works

```
int main() {  
    int value = 137;  
    void becomeWealthy(int netWorth) {  
        netWorth = 10000000000;  
    }  
}
```

137

137
netWorth

How it Works

```
int main() {
```

```
    int value = 127;
```

137

```
    void becomeWealthy(int netWorth) {
```

```
        netWorth = 10000000000;
```

kaching!

```
    }
```

netWorth

```
}
```

How it Works

```
int main() {
```

```
    int value = 127;
```

137

```
    void becomeWealthy(int netWorth) {
```

```
        netWorth = 10000000000;
```

kaching!

```
    }
```

netWorth

```
}
```

How it Works

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

137

value

Parameter Passing in C++

- By default, in C++, parameters are passed by value.

```
/* This function gets a copy of the integer passed
 * into it, so we only change our local copy. The
 * caller won't see any changes.
 */
void byValue(int number) {
    number = 137;
}
```

- You can place an ampersand after the type name to take the parameter by reference.

```
/* This function takes its argument by reference, so
 * when the function returns, the int passed in will have
 * been permanently changed.
 */
void byReference(int& number) {
    number = 137;
}
```

How it Works Now

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

How it Works Now

```
int main() {  
    int value = 137;  
    becomewealthy(value);  
    cout << value << endl;  
    return 0;  
}
```


How it Works Now

```
int main() {  
    int value = 137;  
    becomewealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

137

value

How it Works Now

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

137

value

How it Works Now

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
}
```

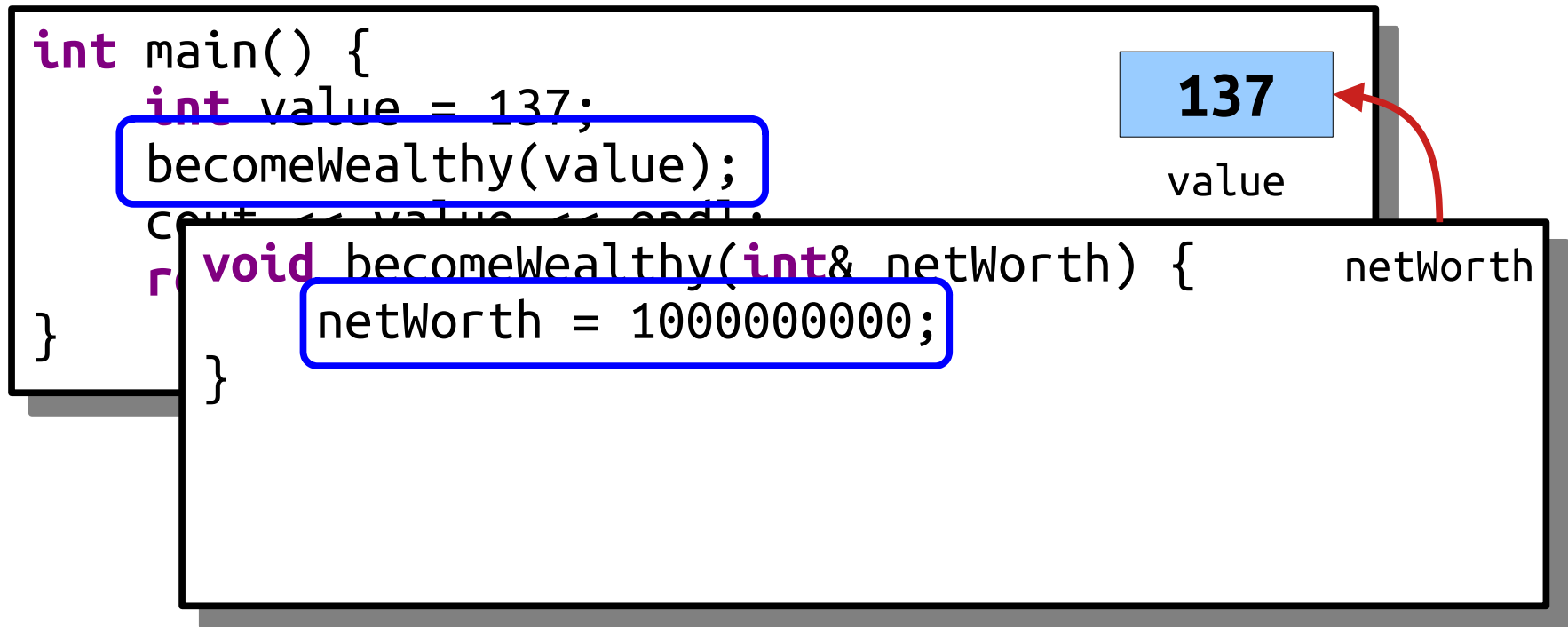
137

value

```
void becomeWealthy(int& netWorth) {  
    netWorth = 10000000000;  
}
```

netWorth

How it Works Now



How it Works Now

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
}  
  
void becomeWealthy(int& netWorth) {  
    netWorth = 10000000000;  
}
```

kaching!

value

netWorth

The diagram illustrates a C++ program where a variable is passed by reference to a function. In the `main` function, an integer `value` is initialized to 137. It is then passed to the `becomeWealthy` function. The function signature is `void becomeWealthy(int& netWorth)`, where `int&` indicates a reference. Inside the function, `netWorth` is assigned the value 10,000,000,000. A blue box labeled "kaching!" with a red arrow pointing to the `value` variable in `main` indicates that the original variable's value has been changed by the function call. The `cout` statement in `main` will output the updated value of `value`.

How it Works Now

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
}
```

kaching!

value

```
void becomeWealthy(int& netWorth) {  
    netWorth = 10000000000;  
}
```

netWorth

How it Works Now

```
int main() {  
    int value = 137;  
    becomeWealthy(value);  
    cout << value << endl;  
    return 0;  
}
```

kaching!

value

Make a Prediction!

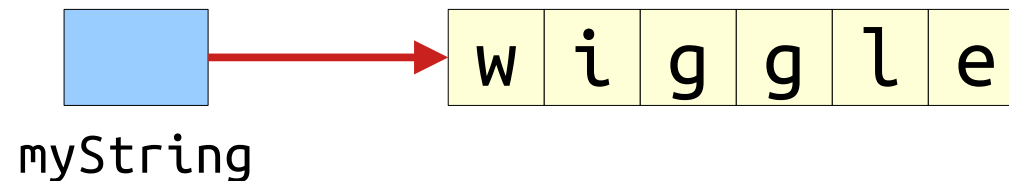
- Look over this piece of C++ code:

```
void gollyGee(string text) {  
    text[0] = 'g';  
}  
  
int main() {  
    string message = "wiggle";  
    gollyGee(message);  
    cout << message << endl; // <-- Here  
    return 0;  
}
```

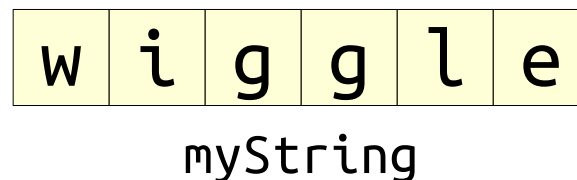
- What do you think will get printed at the indicated point?

Strings in C++

- In Python, Java, and JavaScript, string variables are not the strings themselves. They're pointers to those strings.



- In C++, a variable of type string is an actual, concrete, honest-to-goodness string.



How it Works

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

How it Works

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

How it Works

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

wiggles

message

How it Works

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

wiggles

message

How it Works

```
int main() {  
    string message = "golly gee";  
    gollyGee(message);  
}
```

wiggle

```
void gollyGee(string text) {  
    text[0] = 'g';  
}
```

wiggle

text

How it Works

```
int main() {  
    string message = "hello";  
    void gollyGee(string text) {  
        text[0] = 'g';  
    }  
}
```

wiggle

wiggle

text

How it Works

```
int main() {  
    string message = "wiggles";  
    void gollyGee(string text) {  
        text[0] = 'g';  
    }  
}
```

wiggles

giggle

text

How it Works

```
int main() {  
    string message = "giggle";  
    void gollyGee(string text) {  
        text[0] = 'g';  
    }  
}
```

wiggle

giggle

text

How it Works

```
int main() {  
    string message = "wiggle";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

wiggle

value

Adding An Ampersand

How it Works Now

```
int main() {  
    string message = "wiggle";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

How it Works Now

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

How it Works Now

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

wiggles

message

How it Works Now

```
int main() {  
    string message = "wiggle";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

wiggle

message

How it Works Now

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
}
```

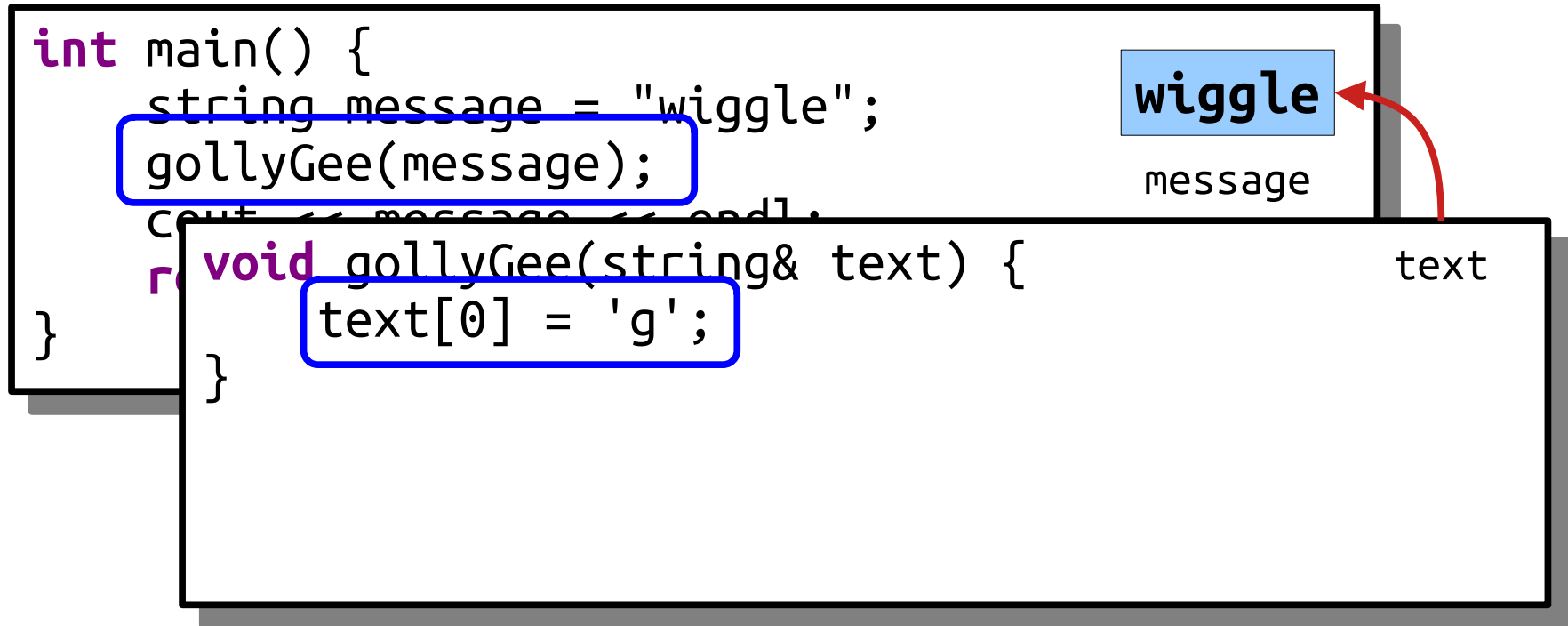
wiggles

message

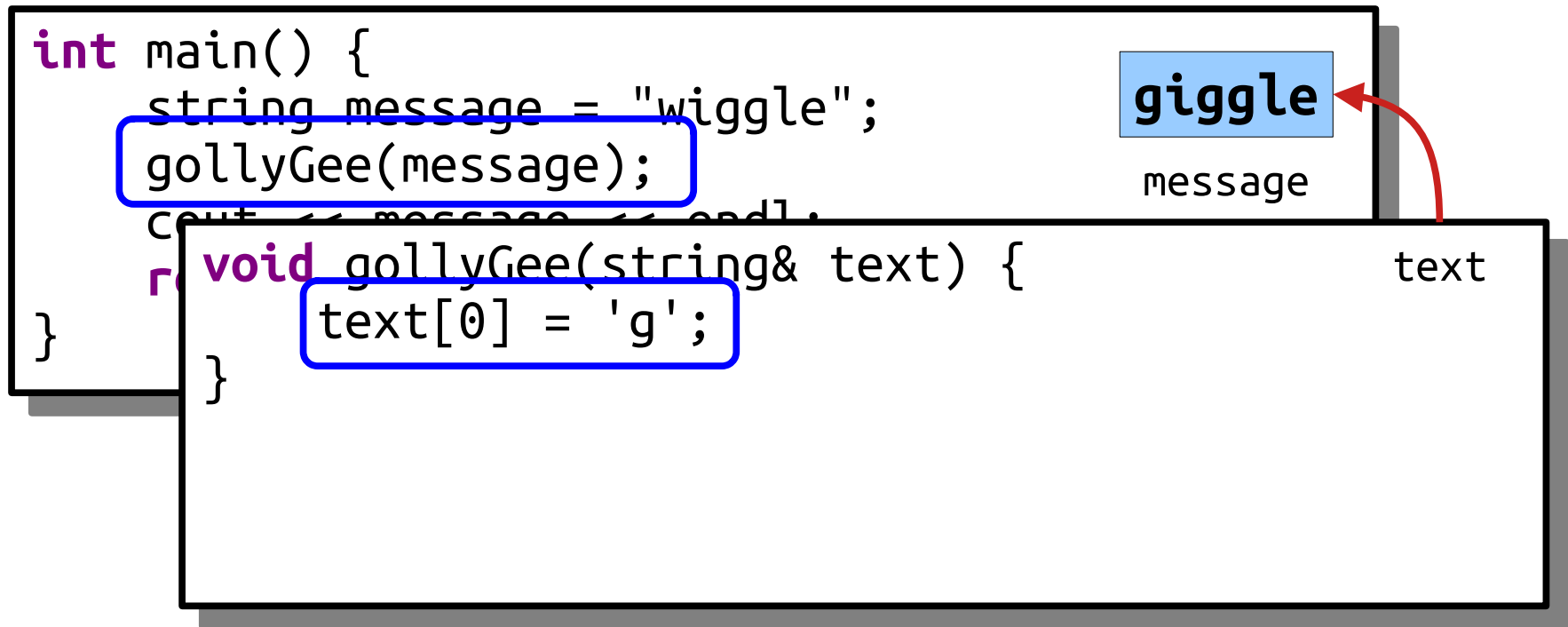
```
void gollyGee(string& text) {  
    text[0] = 'g';  
}
```

text

How it Works Now



How it Works Now



How it Works Now

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
}
```

giggle

message

```
void gollyGee(string& text) {  
    text[0] = 'g';  
}
```

text

How it Works Now

```
int main() {  
    string message = "wiggles";  
    gollyGee(message);  
    cout << message << endl;  
    return 0;  
}
```

giggle

message

A Question of Speed

- When working with strings, pass-by-value is slower than pass-by-reference because of the cost of copying the string.

I		a	m		h	a	p	p	y		t	o		j	o	i	n	
---	--	---	---	--	---	---	---	---	---	--	---	---	--	---	---	---	---	--

 ...

- ***General principle:*** When passing a string into a function, use pass-by-reference unless you actually want a copy of the string.

Do You Trust Me?

- Suppose you've written the next Great American Novel and the single, sole copy is stored in the variable

```
string myMasterpiece;
```

- You see a function with this signature:

```
void totallyNotSketchy(string& text);
```

- Would you make this call?

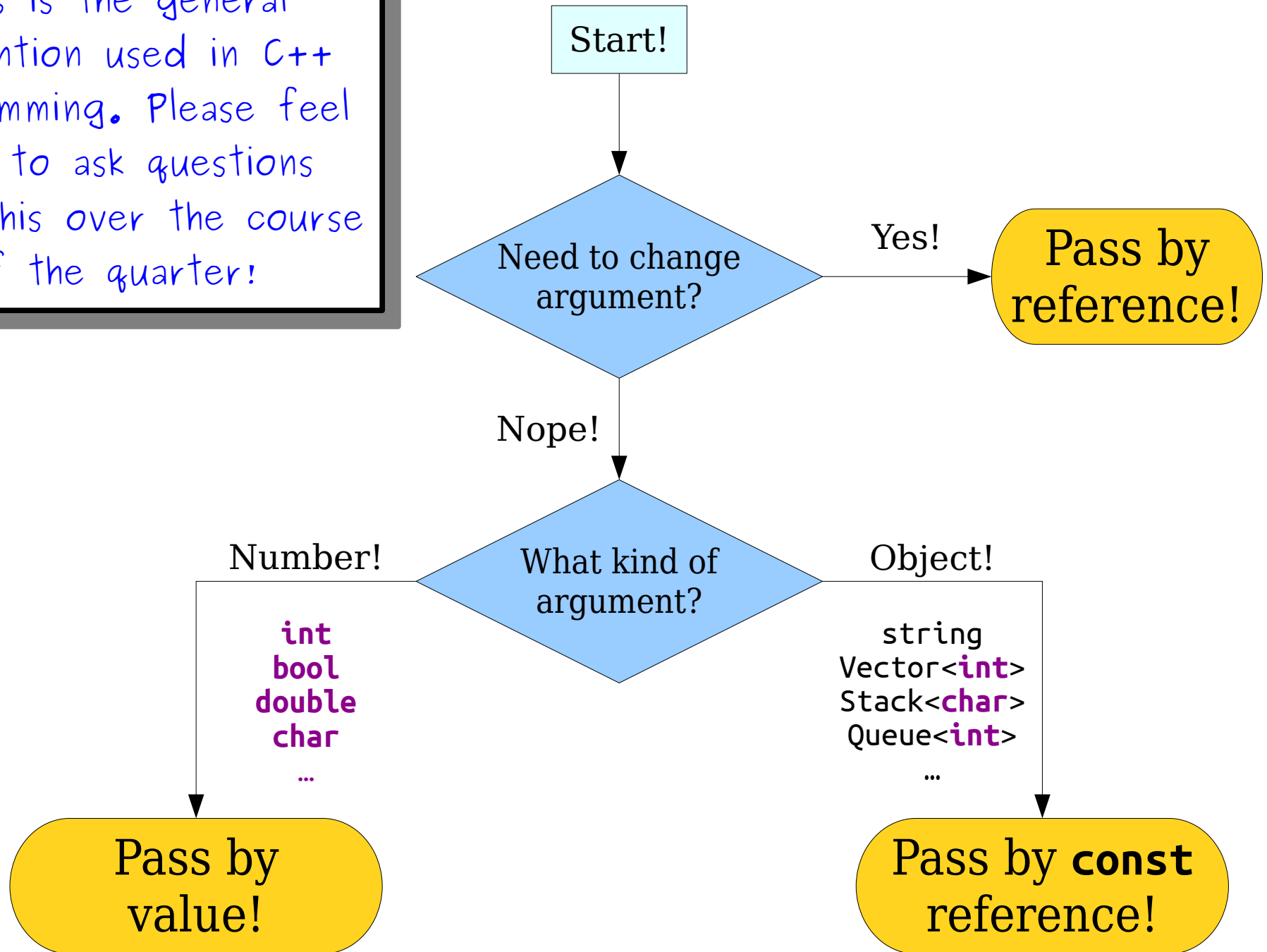
```
totallyNotSketchy(myMasterpiece);
```

Pass-by-const-Reference

- If you want to look at, but not modify, a function parameter, pass it by ***const reference***:
 - The “by reference” part avoids a copy.
 - The “**const**” (constant) part means that the function can’t change that argument.
- For example:

```
void proofreadLongEssay(const string& essay) {  
    /* can read, but not change, the essay. */  
}
```

This is the general convention used in C++ programming. Please feel free to ask questions about this over the course of the quarter!



Container Types

Container Types

- A ***collection class*** (also called an ***abstract data type*** or ***container class***) is a data type used to store and organize data in some form.
 - These are things like arrays, lists, maps, dictionaries, etc.
- Our next three lectures exploring collections and how to use them appropriately.
- Later, we'll analyze their efficiencies. For now, let's just focus on how to use them.

Vector

Vector

- A **Vector** is a collection class representing a list of things.
- It's similar to Java's ArrayList, JavaScript's arrays, and Python's lists.
- To make a Vector, use this syntax:

`Vector<type> name;`
- All elements of a Vector have to have the same type. You specify that type by placing it in <angle brackets> after the word Vector.

Vector in Action

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]

```

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

```

//      JavaScript Version
let v = [1, 3, 7];

v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);

```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]

```

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

```

//      JavaScript Version
let v = [1, 3, 7];

v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);

```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;
cout << v[0] << endl;
cout << v[v.size() - 1] << endl;
Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);
v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)
print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

```

Note the use of curly braces rather than square brackets here.

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

```

let v = [1, 3, 7];

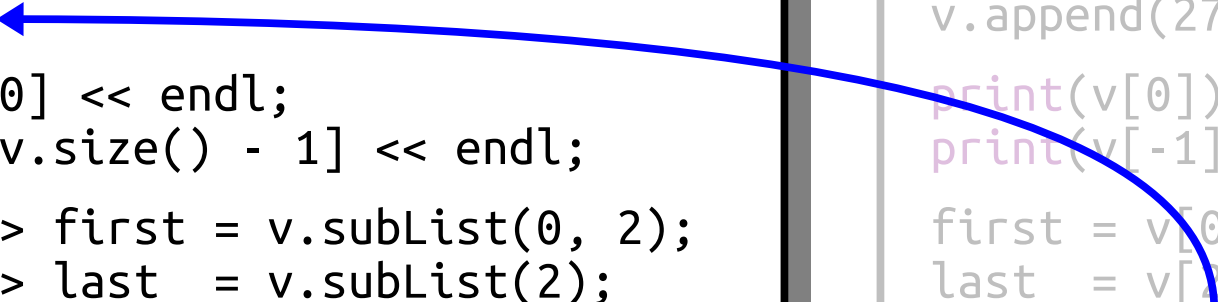
v.push(271);
console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);

```



```
/*      Stanford C++ Version      */  
Vector<int> v = { 1, 3, 7 };  
  
v += 271;   
cout << v[0] << endl;  
cout << v[v.size() - 1] << endl;  
Vector<int> first = v.subList(0, 2);  
Vector<int> last  = v.subList(2);  
v.remove(0);
```

```
"""      Python Version      """  
v = [1, 3, 7]  
  
v.append(271)  
print(v[0])  
print(v[-1])  
first = v[0:2]  
last  = v[2:]
```

We append elements
using the += operator.

```
/*      Java Version      */  
List<> v = new ArrayList<Integer>();  
v.add(1); v.add(3); v.add(7);  
  
v.add(271);  
System.out.println(v.get(0));  
System.out.println(v.get(v.size()-1));  
List<Integer> first = v.subList(0, 2);  
List<Integer> last  = v.subList(2);  
v.remove(0);
```

```
//      JavaScript Version  
let v = [1, 3, 7];  
  
v.push(271);  
console.log(v[0]);  
console.log(v[v.length - 1]);  
let first = v.slice(0, 2);  
let last  = v.slice(2);  
v.splice(0, 0);
```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

```

We select individual elements out of a Vector using square brackets. Everything is zero-indexed.

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

```

v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);

```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;
Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);
v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

```

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);
v.remove(0);

```

C++ doesn't support negative array indices to mean "count from the back." We have to do some math to find the index of the last element.

We use the syntax `v.size()` to get the length of a **Vector**.

```

let last = v.slice(2);
v.splice(0, 0);

```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);
v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]

```

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

The `subList` member function is used to get a subrange of the `subList`. Here, `first` will be the first two elements of the `Vector`, and `last` will be the list starting at position 2.

```

v.splice(0, 0);

```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

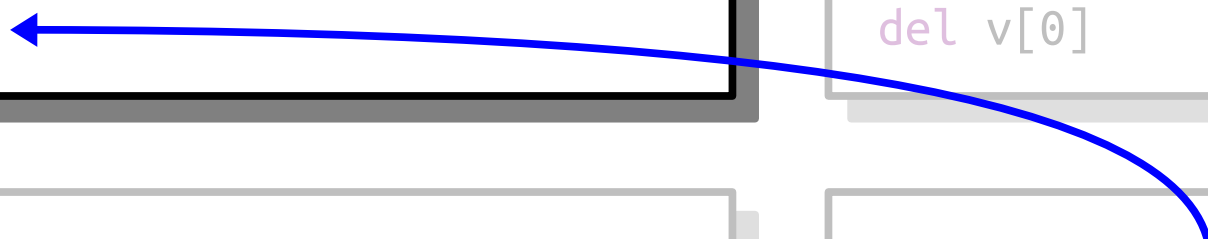
v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);

```



```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]

```

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

We can use the `remove` member function to remove the element at a given index.

```

//      JavaScript Version
v = [1, 3, 7];
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);

```

```

/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };

v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);

```

```

"""      Python Version      """
v = [1, 3, 7]

v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]

```

```

/*      Java Version      */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);

```

```

//      JavaScript Version
let v = [1, 3, 7];

v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);

```



```

/*      Stanford C++ Version      */
Vector<string> v = { "A", "B", "C" };

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}

```

```

"""      Python Version      """
v = ["A", "B", "C"]

# Counting for loop.
for i in range(len(v)):
    print(v[i])

# Range-based for loop.
for elem in v:
    print(elem)

```

```

/*      Java Version      */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}

```

```

//      JavaScript Version
let v = ["A", "B", "C"];

// Counting for loop.
for (let i in v) {
    console.log(v[i]);
}

// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}

```



```

/*          Stanford C++ Version          */
Vector<string> v = { "A", "B", "C" };

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}

```

```

"""          Python Version          """
v = ["A", "B", "C"]

# Counting for loop.
for i in range(len(v)):
    print(v[i])

# Range-based for loop.
for elem in v:
    print(elem)

```

```

/*          Java Version          */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}

```

```

//          JavaScript Version
let v = ["A", "B", "C"];

// Counting for loop.
for (let i in v) {
    console.log(v[i]);
}

// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}

```

```
/*      Stanford C++ Version      */  
Vector<string> v = { "A", "B", "C" };
```

```
/* Counting for loop. */  
for (int i = 0; i < v.size(); i++) {  
    cout << v[i] << endl;  
}
```

```
/* Range-based for loop. */  
for (string elem: v) {  
    cout << elem << endl;  
}
```

```
"""      Python Version      """  
v = ["A", "B", "C"]
```

```
# Counting for loop.  
for i in range(len(v)):  
    print(v[i])
```

```
# Range-based for loop.  
for elem in v:  
    print(elem)
```

```
/*      Java Version      */  
List<> v = new ArrayList<String>();  
v.add("A"); v.add("B"); v.add("C");
```

```
/* Counting for loop. */  
for (int i = 0; i < v.size(); i++) {  
    System.out.println(v[i]);  
}
```

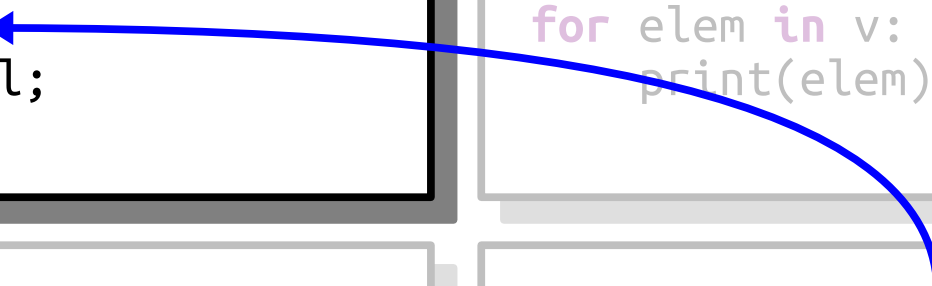
```
/* Range-based for loop. */  
for (String elem: v) {  
    System.out.println(elem);  
}
```

We can iterate over the elements of a Vector by counting upward from 0 (inclusive) to its size (exclusive) and accessing each element.

```
/*      Stanford C++ Version      */
Vector<string> v = { "A", "B", "C" };

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
```



```
"""      Python Version      """
v = ["A", "B", "C"]

# Counting for loop.
for i in range(len(v)):
    print(v[i])

# Range-based for loop.
for elem in v:
    print(elem)
```

```
/*      Java Version      */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}
```

We can also use this loop structure, which visits each element of the vector in the order in which they appear.

```
// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}
```

```

/*      Stanford C++ Version      */
Vector<string> v = { "A", "B", "C" };

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}

```

```

"""      Python Version      """
v = ["A", "B", "C"]

# Counting for loop.
for i in range(len(v)):
    print(v[i])

# Range-based for loop.
for elem in v:
    print(elem)

```

```

/*      Java Version      */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}

```

```

//      JavaScript Version
let v = ["A", "B", "C"];

// Counting for loop.
for (let i in v) {
    console.log(v[i]);
}

// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}

```

To read more about the Vector and how to use it, check out the

Stanford C++ Library Documentation

up on the course website.

Make a Prediction!

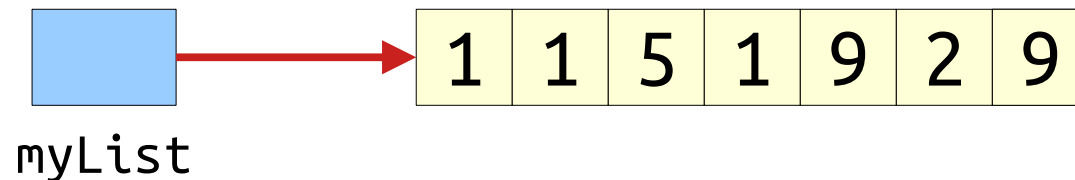
- Look over this piece of C++ code:

```
void dream(Vector<int> numbers) {  
    numbers[1] = 1963;  
}  
  
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl; // <- - Here  
    return 0;  
}
```

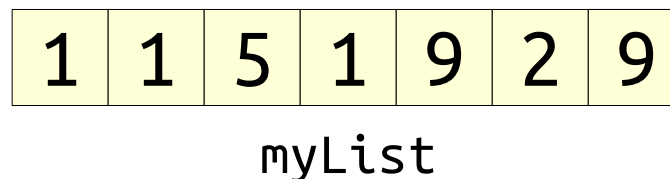
- What do you think will get printed at the indicated point?

Objects in C++

- In Python, Java, and JavaScript, object variables are not the objects themselves. They're pointers to those objects:



- In C++, a variable of object type is an actual, concrete, honest-to-goodness object.



How it Works

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```


How it Works

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

How it Works

1929	1955	1964
------	------	------

values

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

How it Works

1929	1955	1964
------	------	------

values

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

How it Works

1929 1955 1964

1929	1955	1964
------	------	------

numbers

```
void dream(Vector<int> numbers) {  
    numbers[1] = 1963;  
}
```

How it Works

The diagram illustrates a C++ function named `dream` that takes a `Vector<int>` parameter named `numbers`. The function body contains the line `numbers[1] = 1963;`, which is highlighted with a blue box. Above the code, a table represents the state of the `numbers` vector. The table has three columns, with the first two containing the values 1929 and 1955, and the third containing 1964. The label `numbers` is positioned below the table. The code snippet is enclosed in a box with a purple border, and the entire diagram is set against a background of overlapping light gray rectangles.

1929	1955	1964
------	------	------

numbers

```
void dream(Vector<int> numbers) {  
    numbers[1] = 1963;  
}
```

How it Works

The diagram illustrates the execution of a C++ function. A function named `dream` is defined, taking a `Vector<int>` parameter named `numbers`. The function body contains the statement `numbers[1] = 1963;`, where the index `1` is highlighted with a blue box. Above the function, a table represents the state of the `numbers` vector, showing the values `1929`, `1963`, and `1964` at indices `0`, `1`, and `2` respectively. The table is labeled `numbers`.

Index	Value
0	1929
1	1963
2	1964

```
void dream(Vector<int> numbers) {  
    numbers[1] = 1963;  
}
```

How it Works

1929 1963 1964

1929 1963 1964

numbers

```
void dream(Vector<int> numbers) {  
    numbers[1] = 1963;  
}
```

How it Works

1929	1955	1964
------	------	------

values

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```


How it Works Now

How it Works Now

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

How it Works Now

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

How it Works Now

1929	1955	1964
------	------	------

values

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

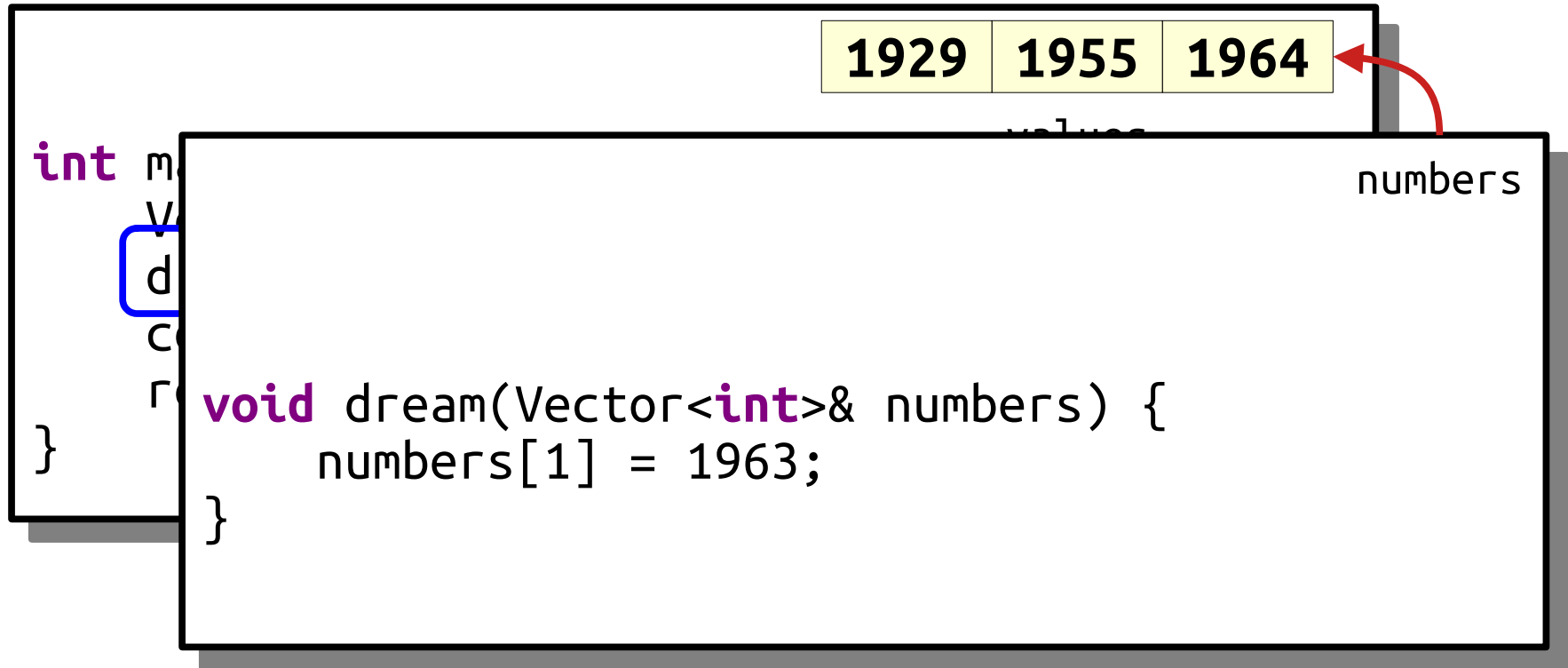
How it Works Now

1929	1955	1964
------	------	------

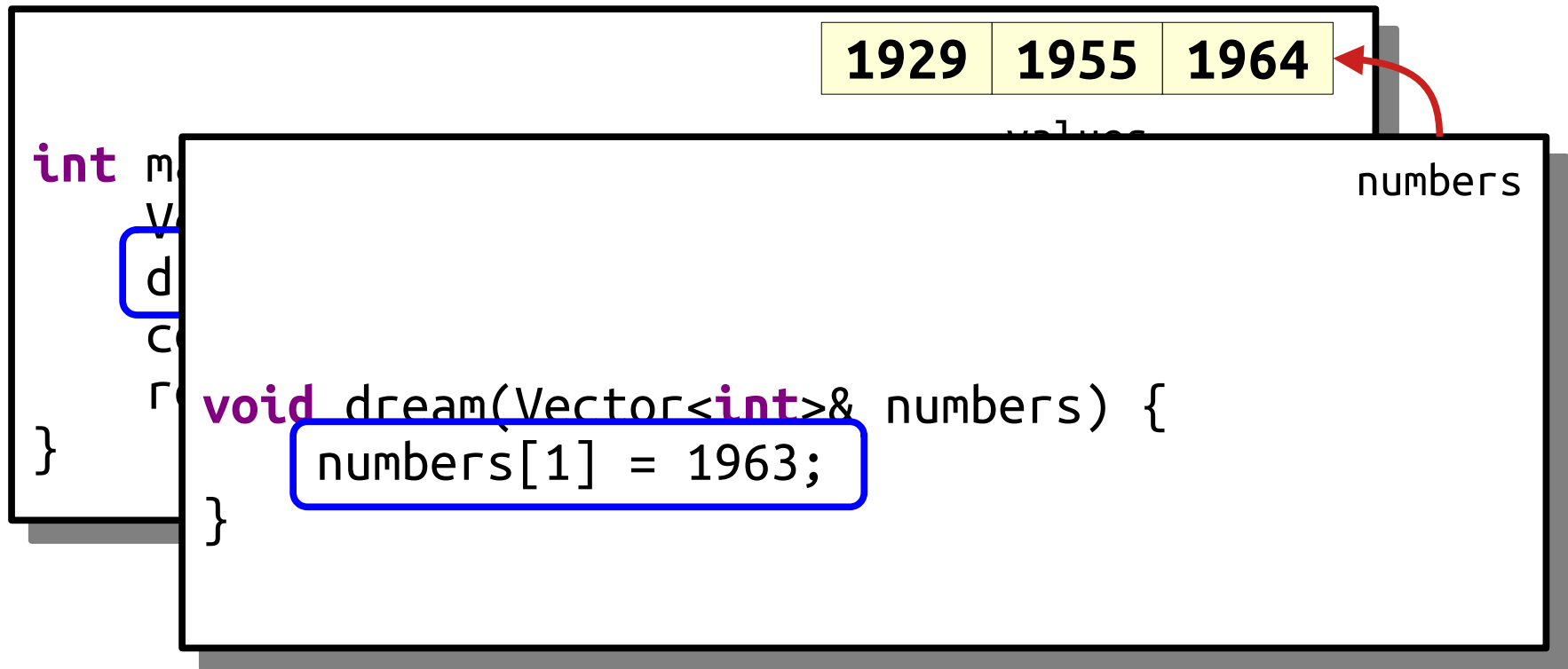
values

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

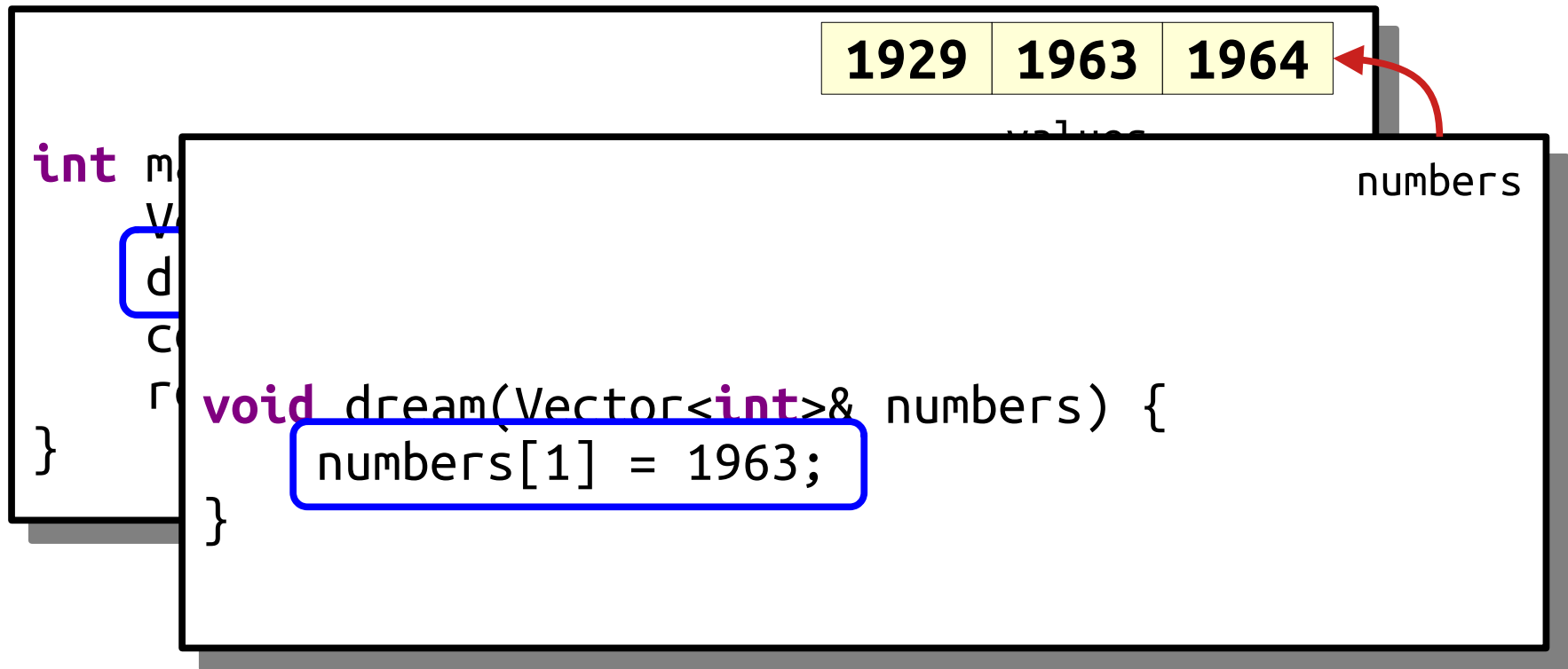
How it Works Now



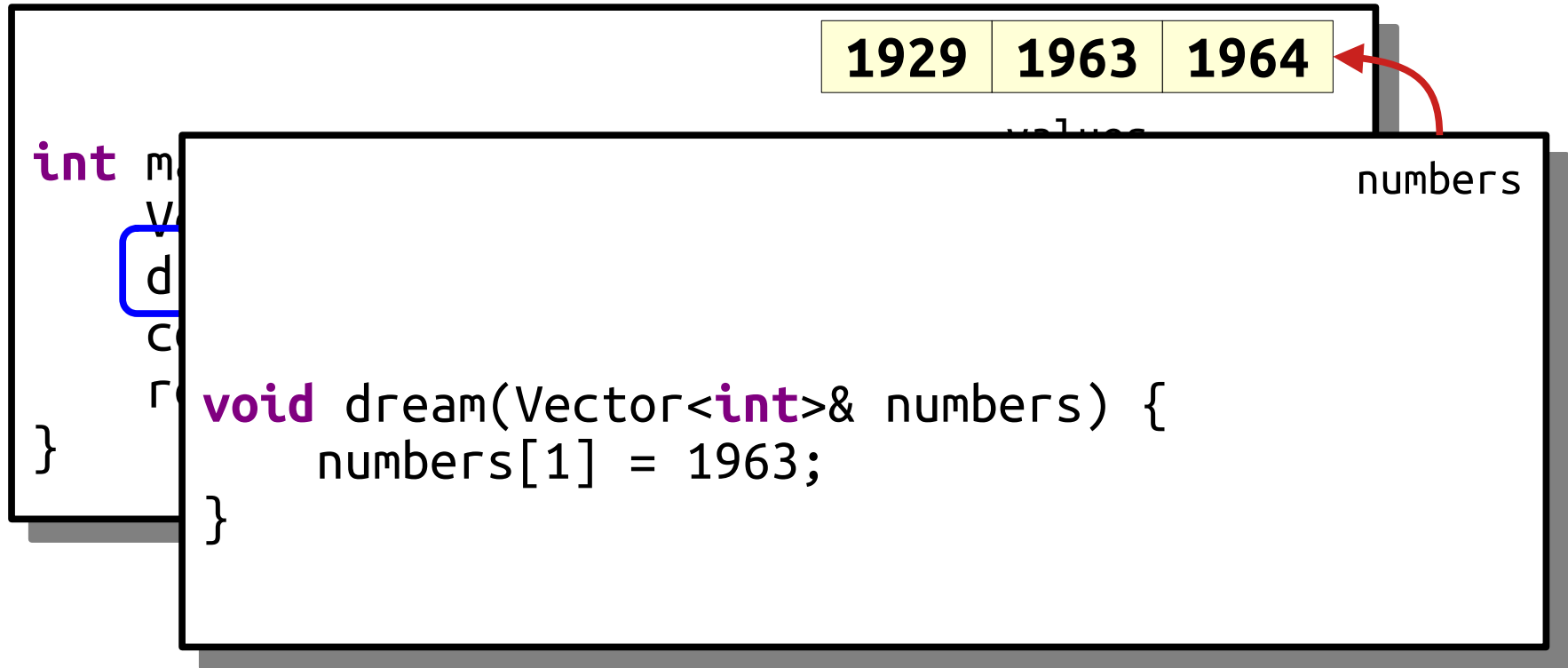
How it Works Now



How it Works Now



How it Works Now



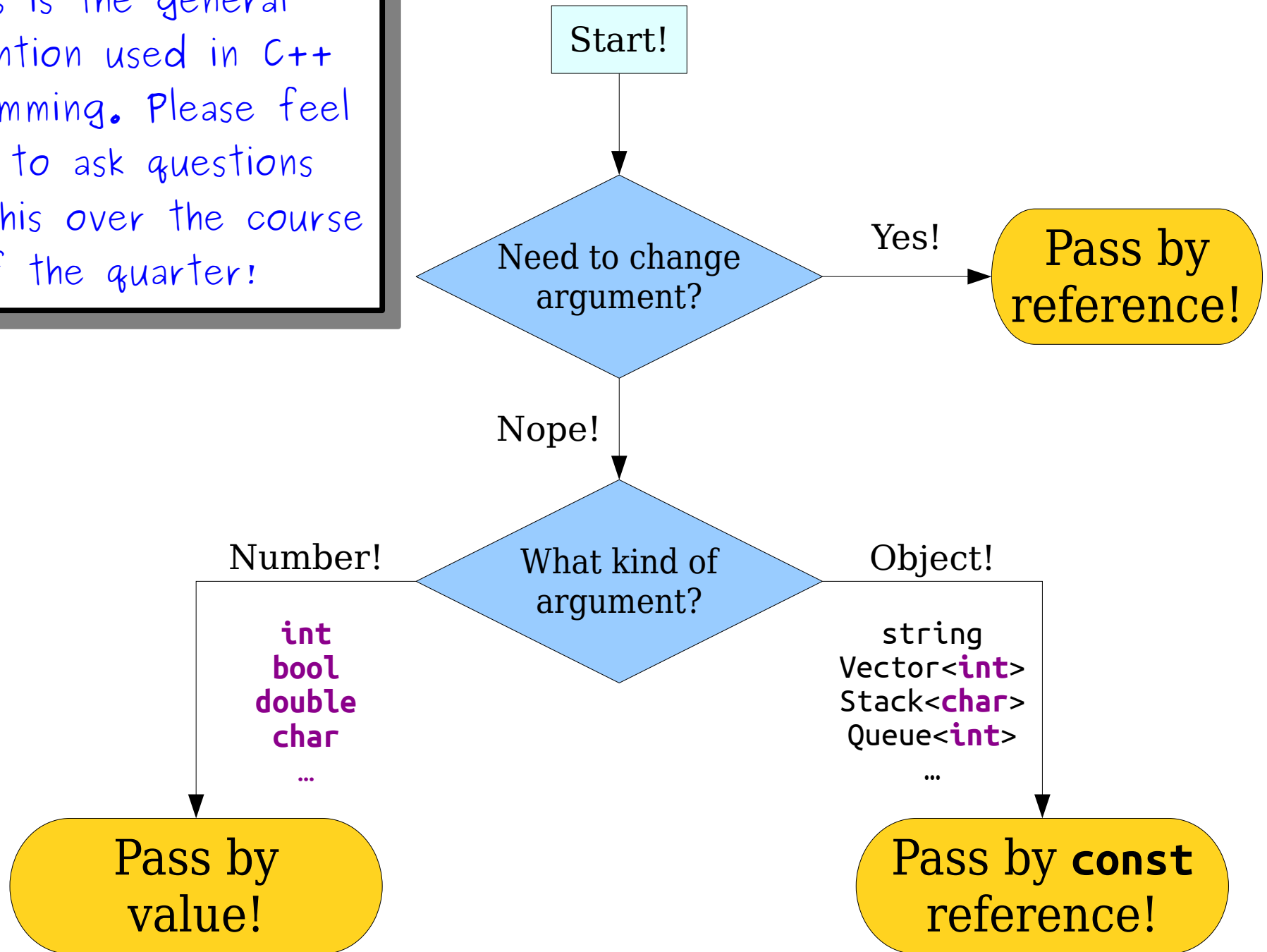
How it Works Now

1929	1963	1964
------	------	------

values

```
int main() {  
    Vector<int> values = { 1929, 1955, 1964 };  
    dream(values);  
    cout << values << endl;  
    return 0;  
}
```

This is the general convention used in C++ programming. Please feel free to ask questions about this over the course of the quarter!



Time-Out for Announcements!

Sections

- Discussion sections start this week!
- Didn't sign up by Sunday at 5PM? The signup link will reopen on Tuesday at 5PM, and you can choose any open section time.
- If your section time doesn't work for you, you can switch into any section with available space starting Tuesday at 5PM. Visit cs198.stanford.edu to do this.
- Still doesn't work for you? Ping Chase!

```
return;
```

Recursion on Vectors

Finding the Largest Number

Finding the Largest Number

- Our goal is to write a function

```
int maxOf(const Vector<int>& numbers);
```

that takes as input a Vector<**int**>, then returns the largest number in the Vector.

- We're going to assume the Vector has at least one element in it; otherwise, it's not possible to return the largest value!
- Let's see how to do this.

Thinking Recursively

```
if (The problem is very simple) {  
    Directly solve the problem.  
    Return the solution.  
}  
else {  
    Split the problem into one or more  
    smaller problems with the same  
    structure as the original.  
    Solve each of those smaller problems.  
    Combine the results to get the overall  
    solution.  
    Return the overall solution.  
}
```

These simple cases
are called *base*
cases.

These are the
recursive cases.

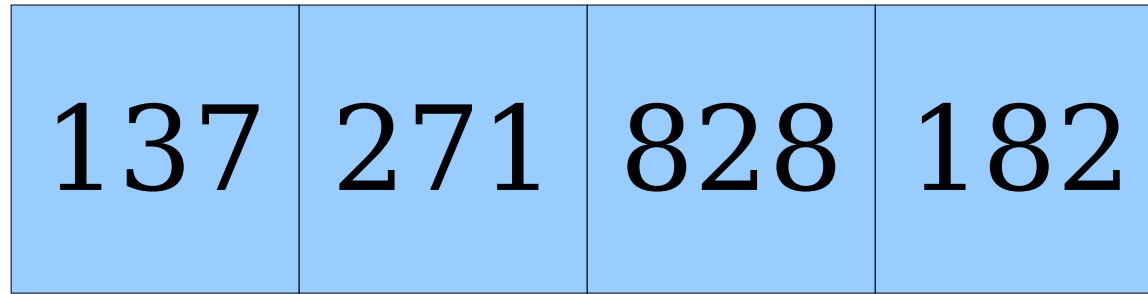
1	2	5	8
---	---	---	---

1	2	5	8
---	---	---	---

I	B	E	X
---	---	---	---

I	B	E	X
---	---	---	---

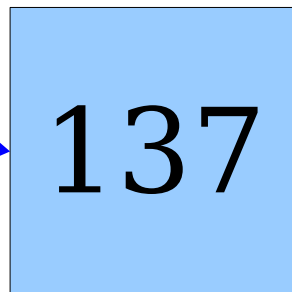
elems



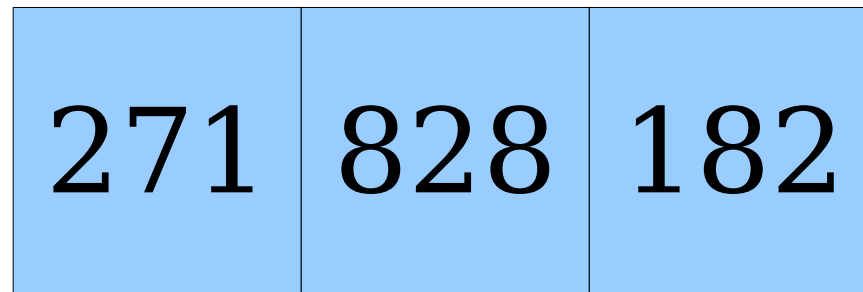
The largest element of
this `Vector<int>` is
either...

... the first
element of the
`Vector<int>`, ...

... or the largest
element in this
`Vector<int>`.



`elems[0]`



`elems.subList(1)`

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 2, 7, 1 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 2, 7, 1 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 2, 7, 1 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

v

2	7	1
---	---	---

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 2, 7, 1 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

v

2	7	1
---	---	---

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 2, 7, 1 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

v

2	7	1
---	---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

first

2

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

first

2

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

Tracing the Recursion

```
i  
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

first 2

rest 7 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

2

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

2

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

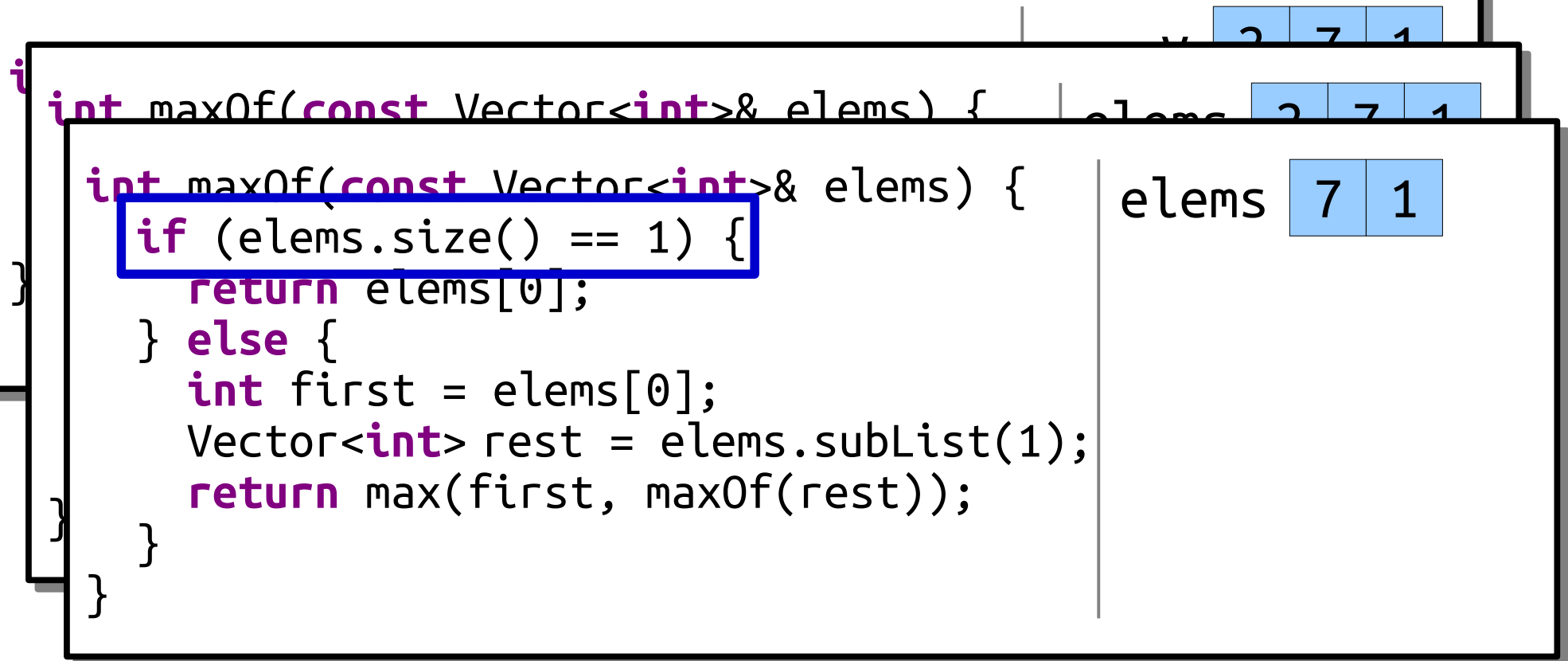
Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

7	1
---	---

Tracing the Recursion



Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

7	1
---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

7	1
---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

7	1
---	---

first

7

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

7	1
---	---

first

7

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems

7	1
---	---

first

7

rest

1

7

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

7

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.subList(1);  
        return max(first, maxOf(rest));  
    }  
}
```

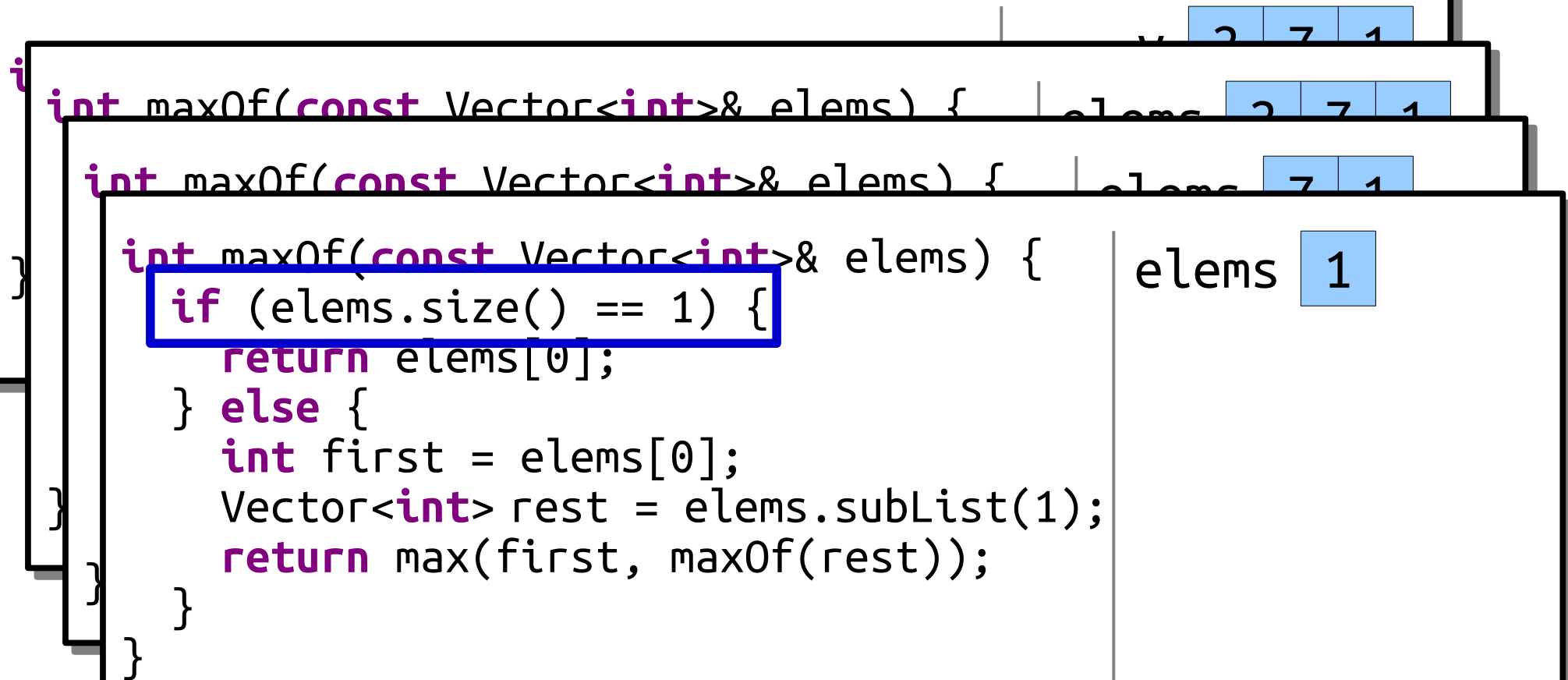
elems 2 7 1

elems 2 7 1

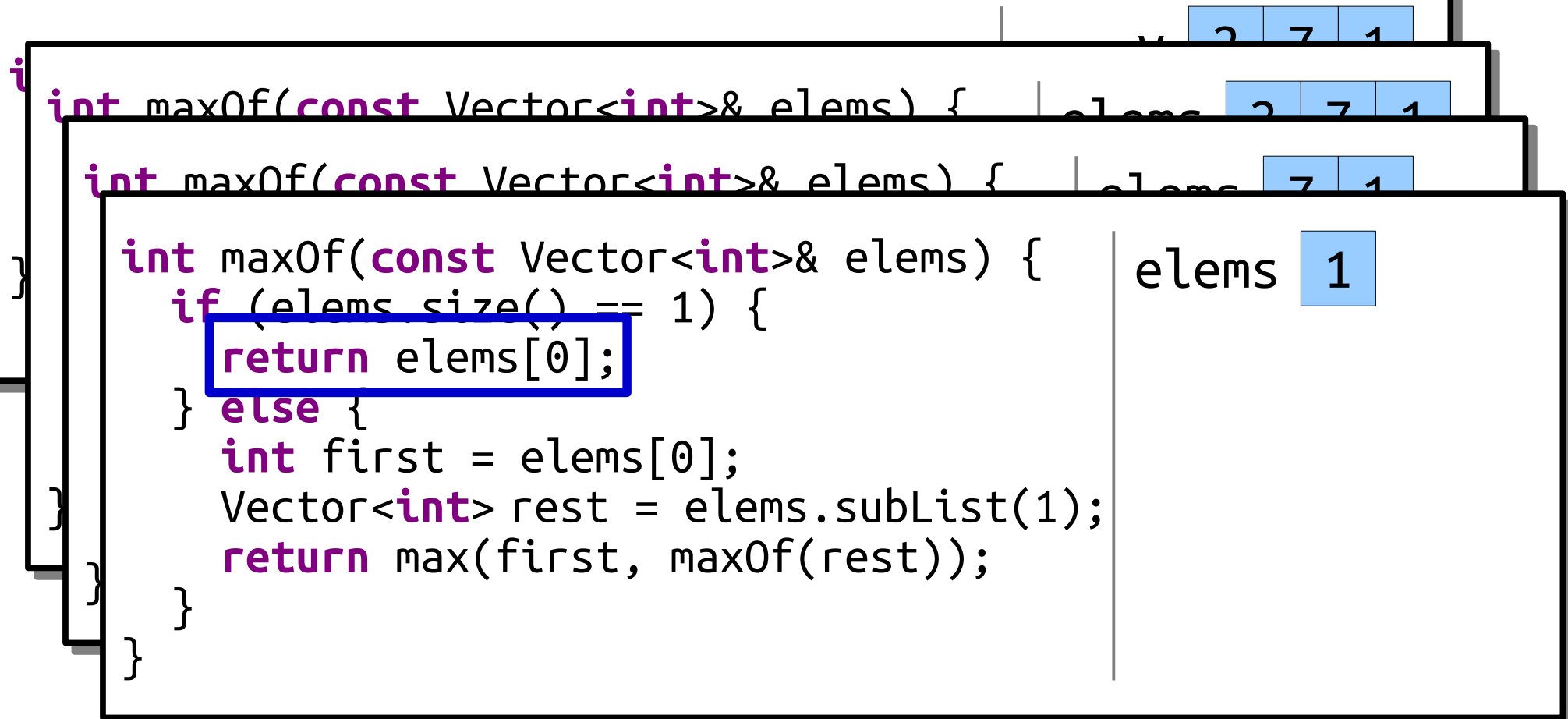
elems 7 1

elems 1

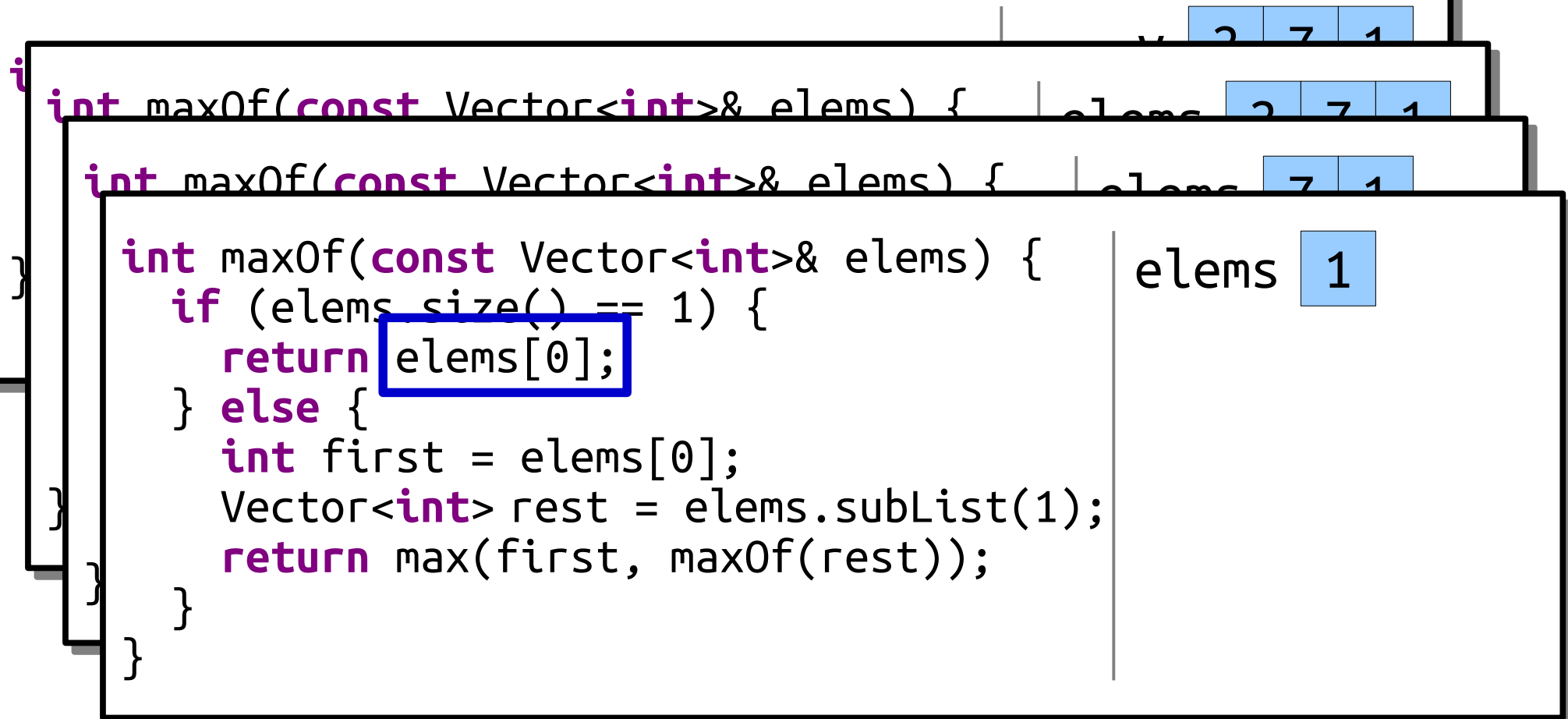
Tracing the Recursion



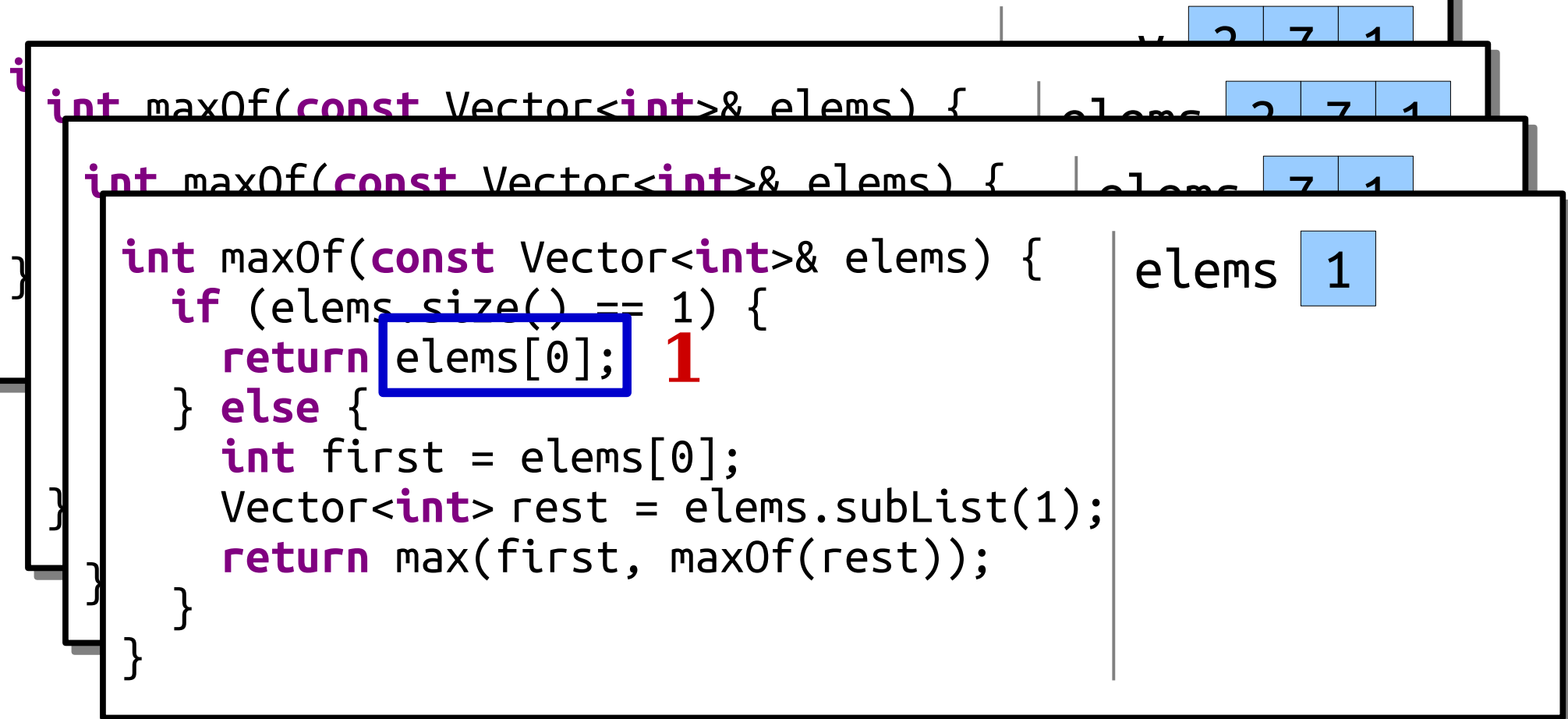
Tracing the Recursion



Tracing the Recursion



Tracing the Recursion



Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

7

1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

7

1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

7

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

7

elems 2 7 1

elems 2 7 1

elems 7 1

first 7

rest 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems.sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

2 7

elems 2 7 1

first 2

rest 7 1

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

2 7

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

7

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int first = elems[0];  
        Vector<int> rest = elems sublist(1);  
        return max(first, maxOf(rest));  
    }  
}
```

7

elems

2	7	1
---	---	---

first

2

rest

7	1
---	---

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 2, 7, 1 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

v

2	7	1
---	---	---

7

Summary from Today

- The `Vector<T>` type in C++ represents a sequence of elements.
- Parameters in C++ are passed by *value* by default. You can change that to use pass by *reference* if you'd like.
- Use pass-by-**const**-reference for objects you don't intend to change.
- Each stack frame from a recursive function gets its own copies of all the local variables.

Your Action Items

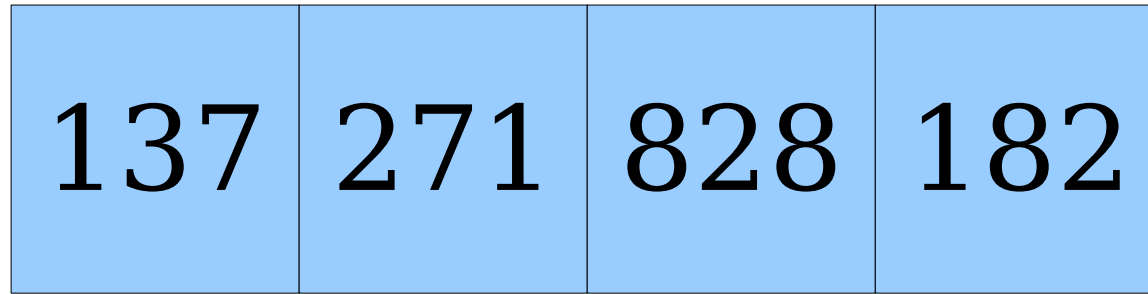
- ***Read Chapter 5.1 of the textbook.***
 - It's all about Vector! There are some goodies there.
- ***Work on Assignment 1.***
 - Aim to complete all three recursion problems by Tuesday evening.
 - Not done by then? Don't worry! Stop by the LaIR to ask questions.
 - Start working on Plotter.
- ***Explore the `maxOf` example.***
 - Tinker and play around with this one. See if you can get very comfortable with how it works.

Next Time

- ***Stacks***
 - How driveways relate to parentheses.
- ***Queues***
 - And a fun application.

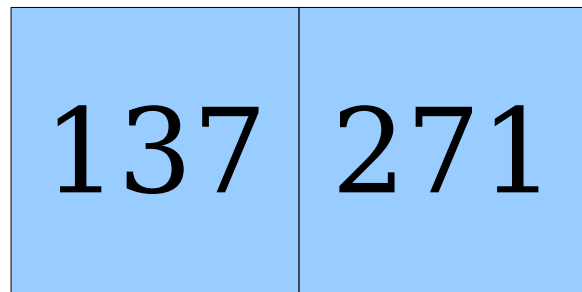
Appendix: Finding the max, another way.

elems



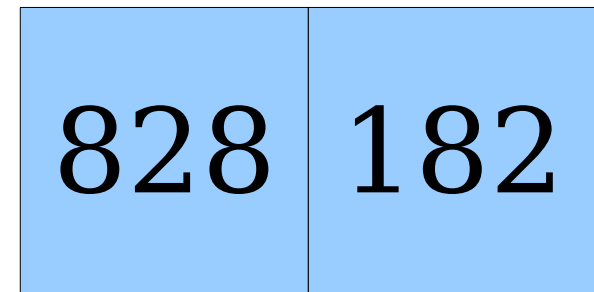
The largest element of
this `Vector<int>` is
either...

... the largest
element in this
`Vector<int>`, ...



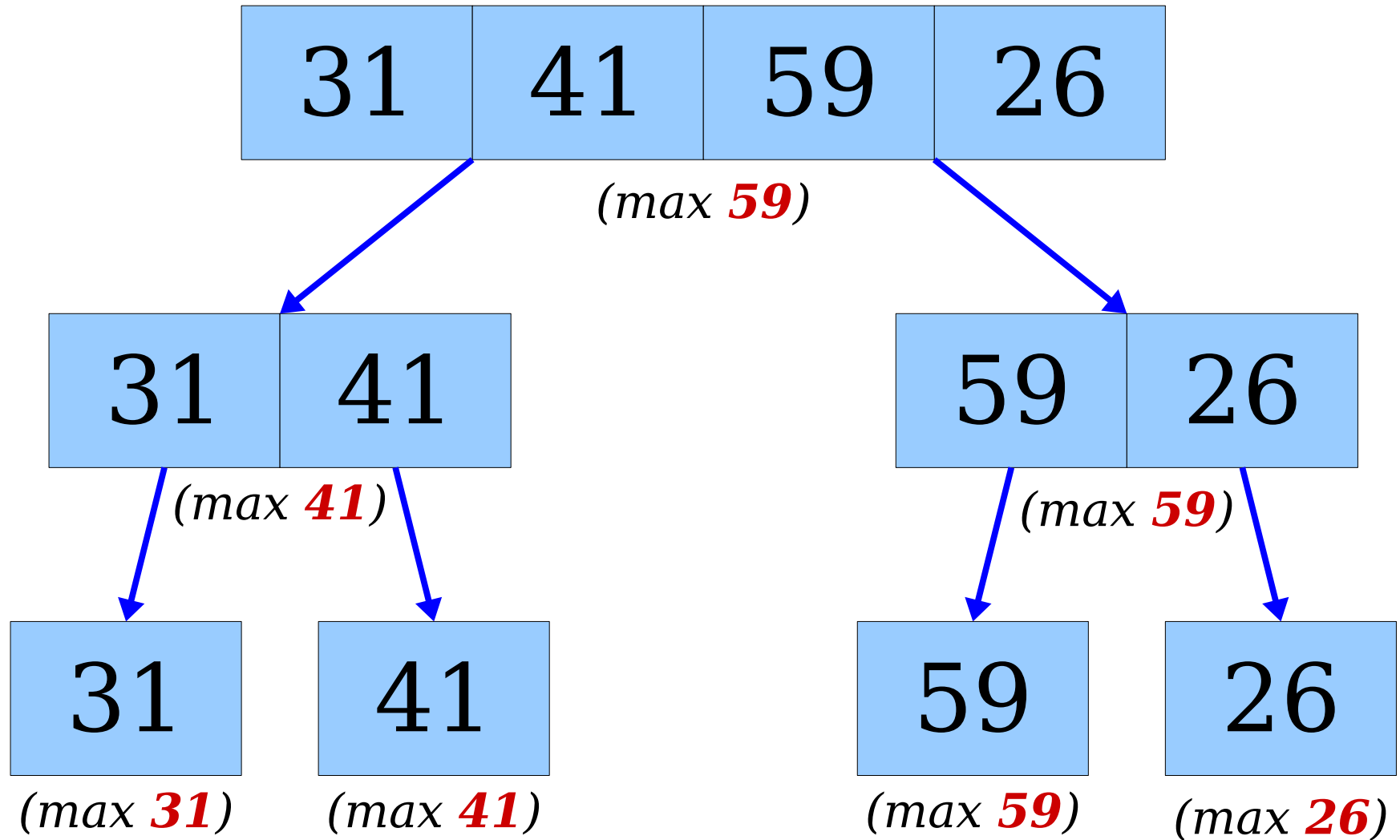
`elems.subList(0, elems.size() / 2)`

... or the largest
element in this
`Vector<int>`.



`elems.subList(elems.size() / 2)`

maxOf as a Tournament



Tracing the Recursion

```
int main() {  
    Vector<int> v = { 31, 41, 59, 26 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

Tracing the Recursion

```
int main() {  
    Vector<int> v = { 31, 41, 59, 26 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

Tracing the Recursion

v

31	41	59	26
----	----	----	----

```
int main() {  
    Vector<int> v = { 31, 41, 59, 26 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

Tracing the Recursion

v

31	41	59	26
----	----	----	----

```
int main() {  
    Vector<int> v = { 31, 41, 59, 26 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```


Tracing the Recursion

v

31	41	59	26
----	----	----	----

```
int main() {  
    Vector<int> v = { 31, 41, 59, 26 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

```
i
int maxOf(const Vector<int>& elems) {
    if (elems.size() == 1) {
        return elems[0];
    } else {
        int half = elems.size() / 2;
        Vector<int> left  = elems.subList(0, half);
        Vector<int> right = elems.subList(half);
        return max(maxOf(left), maxOf(right));
    }
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
i
int maxOf(const Vector<int>& elems) {
    if (elems.size() == 1) {
        return elems[0];
    } else {
        int half = elems.size() / 2;
        Vector<int> left = elems.subList(0, half);
        Vector<int> right = elems.subList(half);
        return max(maxOf(left), maxOf(right));
    }
}
```

v 31 41 59 26

elems 31 41 59 26

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

```
i
int maxOf(const Vector<int>& elems) {
    if (elems.size() == 1) {
        return elems[0];
    } else {
        int half = elems.size() / 2;
        Vector<int> left = elems.subList(0, half);
        Vector<int> right = elems.subList(half);
        return max(maxOf(left), maxOf(right));
    }
}
```

Tracing the Recursion

```
i
int maxOf(const Vector<int>& elems) {
    if (elems.size() == 1) {
        return elems[0];
    } else {
        int half = elems.size() / 2;
        Vector<int> left = elems.subList(0, half);
        Vector<int> right = elems.subList(half);
        return max(maxOf(left), maxOf(right));
    }
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

Tracing the Recursion

```
i
int maxOf(const Vector<int>& elems) {
    if (elems.size() == 1) {
        return elems[0];
    } else {
        int half = elems.size() / 2;
        Vector<int> left = elems.subList(0, half);
        Vector<int> right = elems.subList(half);
        return max(maxOf(left), maxOf(right));
    }
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

Tracing the Recursion

```
i  
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

Tracing the Recursion

```
i
int maxOf(const Vector<int>& elems) {
    if (elems.size() == 1) {
        return elems[0];
    } else {
        int half = elems.size() / 2;
        Vector<int> left = elems.subList(0, half);
        Vector<int> right = elems.subList(half);
        return max(maxOf(left), maxOf(right));
    }
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {
```

```
    if (elems.size() == 1) {
```

```
        return elems[0];
```

```
    } else {
```

```
        int half = elems.size() / 2;
```

```
        Vector<int> left = elems.subList(0, half);
```

```
        Vector<int> right = elems.subList(half);
```

```
        return max(maxOf(left), maxOf(right));
```

```
    }
```

```
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

```
int maxOf(const Vector<int> & elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```


Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 31

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```


Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 31

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 31

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 31

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0]; 31  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

31

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

31

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```


Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 31 41

elems 41

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0]; 41  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

31

41

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

31

41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 31 41

half 1

left 31

right 41

41

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

41

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

41

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```


Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

```
int maxOf(const Vector<int> & elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 59

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 59

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 59

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 59

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0]; 59  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

59

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

elems 59 26

elems 26

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0]; 26  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

59

26

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

59

26

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half, elems.size());  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

elems 59 26

half 1

left 59

right 26

59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

41

59

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v 31 41 59 26

elems 31 41 59 26

half 2

left 31 41

right 59 26

41

59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v	31	41	59	26
elems	31	41	59	26
half	2			
left	31	41		
right			59	26

59

Tracing the Recursion

```
int maxOf(const Vector<int>& elems) {  
    if (elems.size() == 1) {  
        return elems[0];  
    } else {  
        int half = elems.size() / 2;  
        Vector<int> left = elems.subList(0, half);  
        Vector<int> right = elems.subList(half);  
        return max(maxOf(left), maxOf(right));  
    }  
}
```

v	31	41	59	26
elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

59

Tracing the Recursion

v

31	41	59	26
----	----	----	----

```
int main() {  
    Vector<int> v = { 31, 41, 59, 26 };  
    cout << maxOf(v) << endl;  
    return 0;  
}
```

59