

Implementing Abstractions

Part One

Turtles All the Way Down?

- Last time, we implemented a RandomBag on top of our library Vector type.
- But the Vector type is itself a library – what is it layered on top of?
- **Question:** What are the fundamental building blocks provided by the language, and how do we use them to build our own custom classes?

Getting Storage Space

- The Vector, Stack, Queue, etc. all need storage space to put the elements that they store.
- That storage space is allocated using ***dynamic memory allocation***.
- Essentially:
 - You can, at runtime, ask for extra storage space, which C++ will give to you.
 - You can use that storage space however you'd like.
 - You have to explicitly tell the language when you're done using the memory.

Dynamic Allocation Demo

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues 7

```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7


```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr



```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr



```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ":  
    }  
}
```

Because the variable **arr** points to the array, it is called a *pointer*.

numValues

7

arr



```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr



```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

0



```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

0



```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

0

We


```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

0

We

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

1

We

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

1

We

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

1

We

Can

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

1

We

Can

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

2

We

Can

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

2

We

Can

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

2

We

Can

Dance


```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

2

We

Can

Dance

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

3

We

Can

Dance

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

3

We

Can

Dance

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues **7** arr i **3**

We

Can

Dance

If

```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

3

We

Can

Dance

If

```
int main() {  
    int numValues = getInteger("How many lines? ");  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

4

We

Can

Dance

If

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues **7** arr i **4**

We

Can

Dance

If

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues **7** arr i **4**

We Can Dance If We


```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

4

We

Can

Dance

If

We

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

5

We

Can

Dance

If

We

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

5

We

Can

Dance

If

We

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

5

We

Can

Dance

If

We

Want

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

5

We

Can

Dance

If

We

Want

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

6

We

Can

Dance

If

We

Want

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

6

We

Can

Dance

If

We

Want

```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues **7** arr i **6**

We

Can

Dance

If

We

Want

To


```
int main() {
    int numValues = getInteger("How many lines? ");

    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getLine();
    }

    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```

numValues

7

arr

i

7

We

Can

Dance

If

We

Want

To

```
int main() {  
    int numValues = getInteger("How many lines? ");  
  
    string* arr = new string[numValues];  
    for (int i = 0; i < numValues; i++) {  
        arr[i] = getLine();  
    }  
  
    for (int i = 0; i < numValues; i++) {  
        cout << i << ": " << arr[i] << endl;  
    }  
}
```

numValues

7

arr

i

7

We

Can

Dance

If

We

Want

To

Dynamically Allocating Arrays

- First, declare a variable that will point at the newly-allocated array. If the array elements have type T , the pointer will have type T^* .
 - e.g. `int*`, `string*`, `Vector<double>*`
- Then, create a new array with the `new` keyword and assign the pointer to point to it.
- In two separate steps:

```
 $T^*$  arr;  
arr = new  $T$ [size];
```

- Or, in the same line:

```
 $T^*$  arr = new  $T$ [size];
```

Dynamically Allocating Arrays

- C++'s language philosophy prioritizes speed over safety and simplicity.
- The array you get from `new[]` is ***fixed-size***: it can neither grow nor shrink once it's created.
 - The programmer's version of "conservation of mass."
- The array you get from `new[]` has ***no bounds-checking***. Walking off the beginning or end of an array triggers *undefined behavior*.
 - Literally anything can happen: you read back garbage, you crash your program, or you let a hacker take over your computer. Do a search for "buffer overflow" for more details.

Cleaning Up

- When declaring local variables or parameters, C++ will automatically handle memory allocation and deallocation for you.
- When using **new**, you are responsible for deallocating the memory you allocate.
- If you don't, you get a **memory leak**. Your program will never be able to use that memory again.
 - Too many leaks can cause a program to crash - it's important to not leak memory!

Cleaning Up

- You can deallocate memory with the `delete[]` operator:

```
delete[] ptr;
```

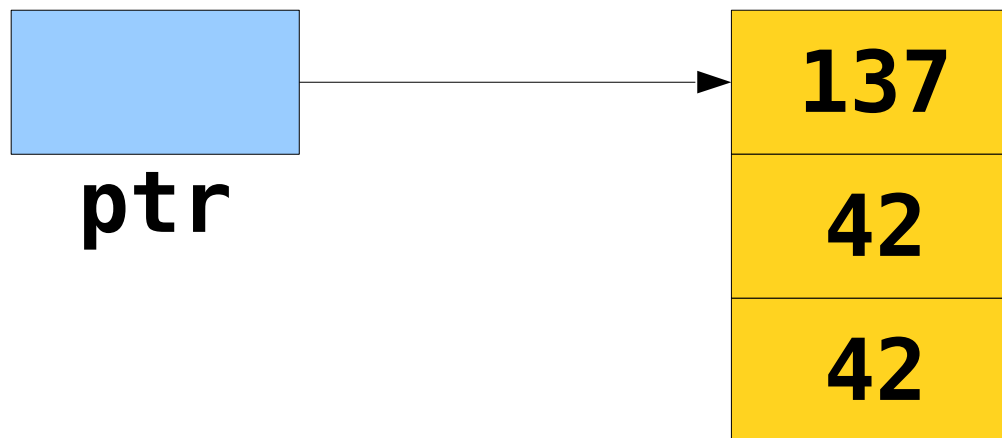
- This destroys the array pointed at by the given pointer, not the pointer itself.

Cleaning Up

- You can deallocate memory with the **delete[]** operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.

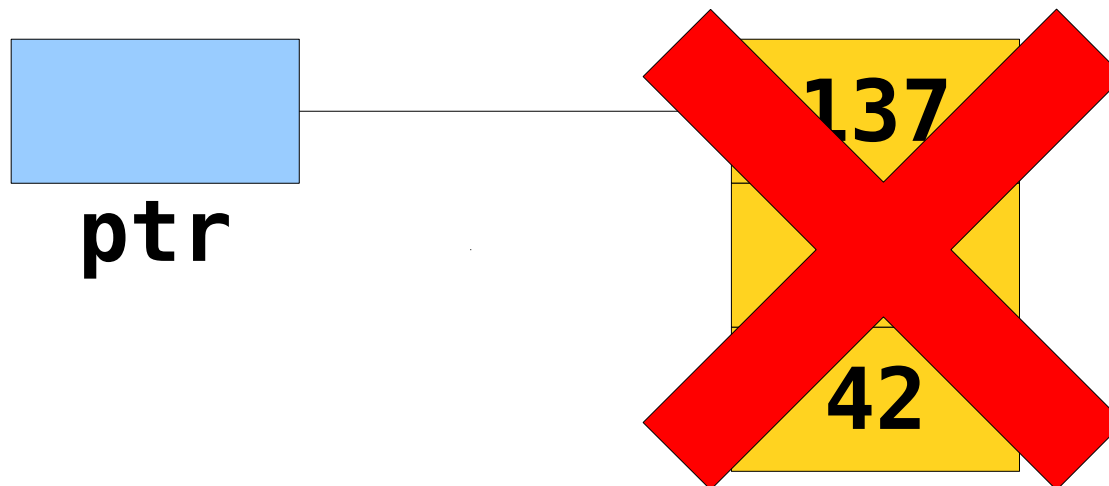


Cleaning Up

- You can deallocate memory with the `delete[]` operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.



Cleaning Up

- You can deallocate memory with the **delete[]** operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.



Cleaning Up

- You can deallocate memory with the `delete[]` operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.



ptr is now a **dangling pointer**. We can reassign it to point somewhere else, but if we try to read from it, it'll do Cruel and Unusual Things!

Time-Out for Announcements!

Midterm Exam

- The midterm exam is next Tuesday, February 21 from 7:00PM – 10:00PM.
 - Location TBA
- Covers topics up through and including big-O notation, plus Assignments 0 – 4.
- Closed-book, closed-computer, limited-note. You get one double-sided sheet of 8.5" × 11" notes when you take the exam. We also provide a library reference sheet.
- We're holding a practice exam **tonight, right here** from 7:00PM – 10:00PM.
 - **You should plan to attend the practice exam unless you have a hard conflict.** The actual exam should not be the first time you write code on paper under time pressure.
- Can't make the exam time? You **must** contact Anton by 5:00PM today.

Assignment 4

- Assignment 4 is due on Friday.
- If you're following our timetable, you should aim to complete Doctors Without Orders, Disaster Planning, and DNA Detective by this evening.
- You should aim to complete the Winning the Presidency part of the assignment by Wednesday evening.
- Please ask questions on Piazza, stop by Keith's or Anton's office hours, or drop by the LaIR if you have questions!

A Humble Plea

- Please feel free to ask questions on Piazza.
- However, if you do, please make sure that the question you're asking hasn't already been answered before - we're getting a lot of duplicate questions.
- That's all!

continue;

Implementing Stack

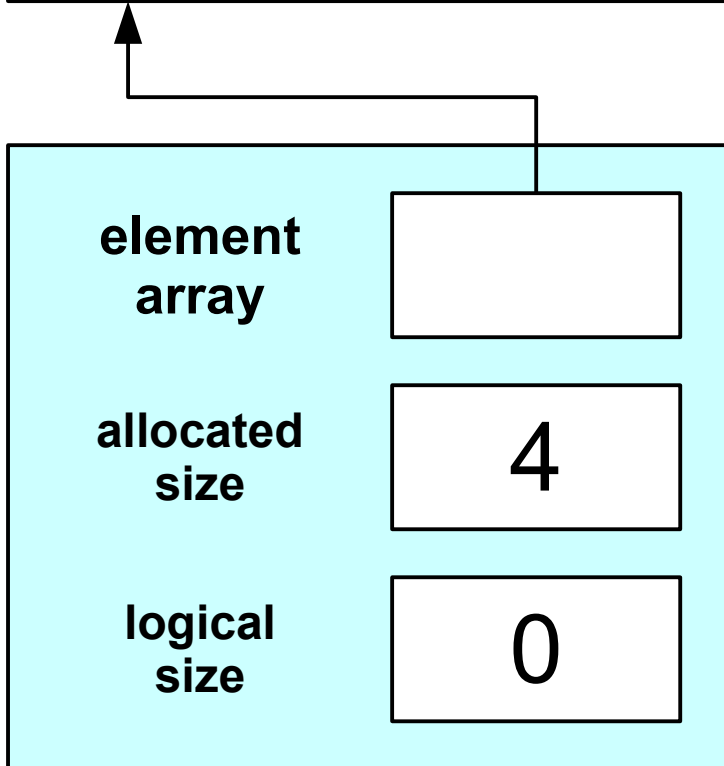
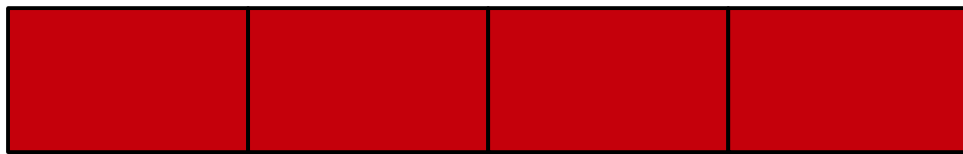
Implementing Stack

- Last time, we saw how to implement RandomBag in terms of Vector.
- We could also implement Stack in terms of Vector.
- What if we wanted to implement the Stack without relying on any other collections?
- Let's build the stack directly!

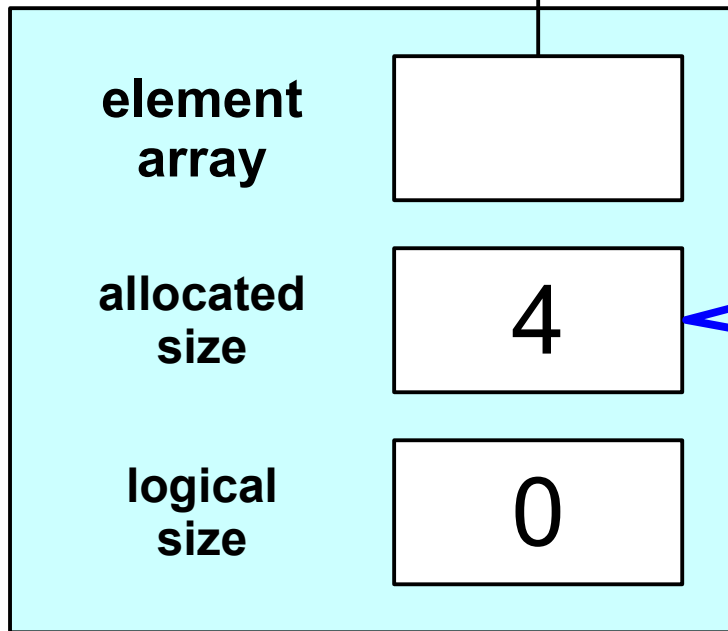
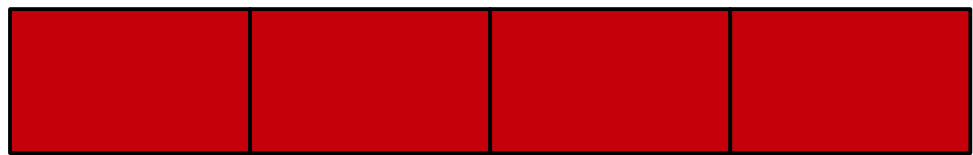
You Gotta Start Somewhere

- Our initial implementation of the stack will be a *bounded* stack with a maximum capacity.
- We'll allocate a fixed amount of storage space for the elements, then write them into the array as they're pushed.
- If we run out of space, we'll report an error.
- Next time, we'll update this code so that we can have a stack without any fixed maximum capacity.

An Initial Idea

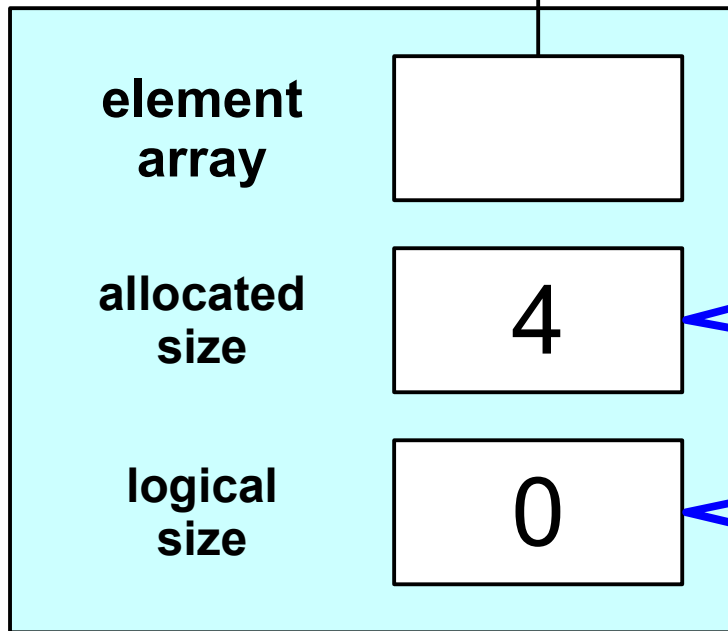
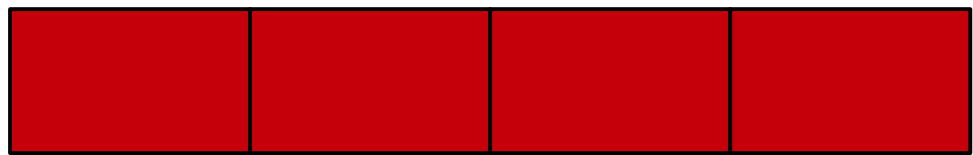


An Initial Idea



The stack's *allocated size* is the number of slots in the array. Remember - arrays in C++ cannot grow or shrink.

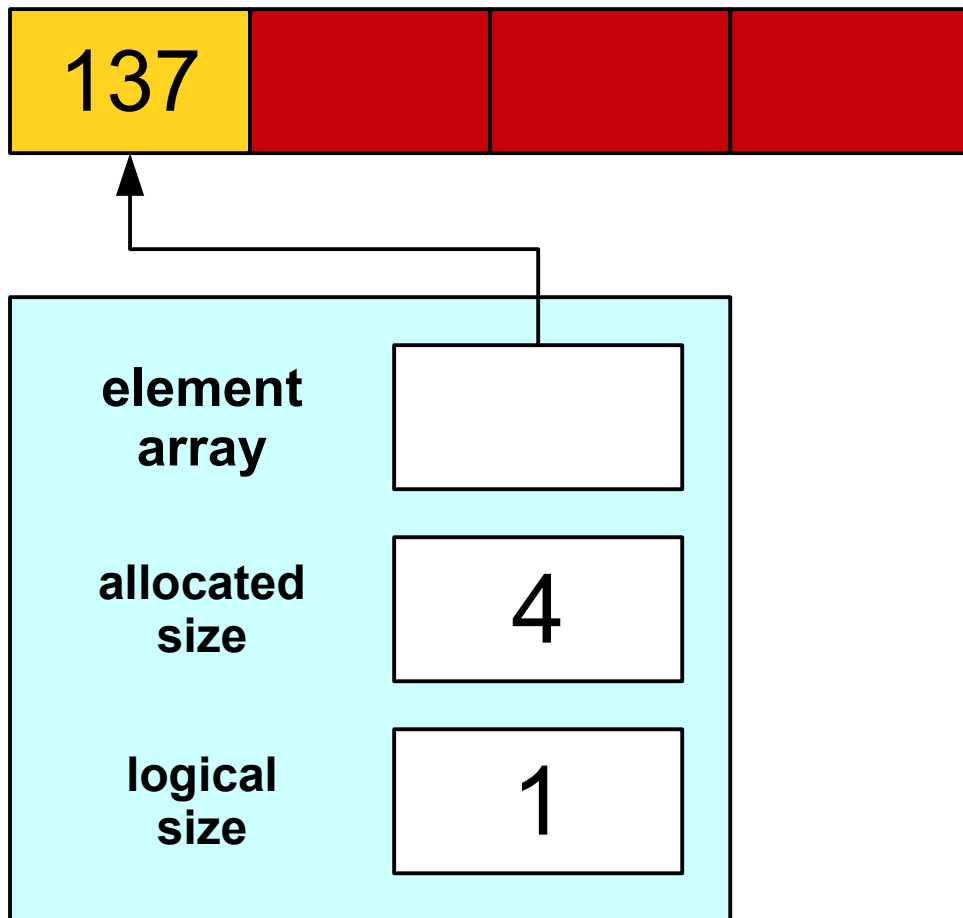
An Initial Idea



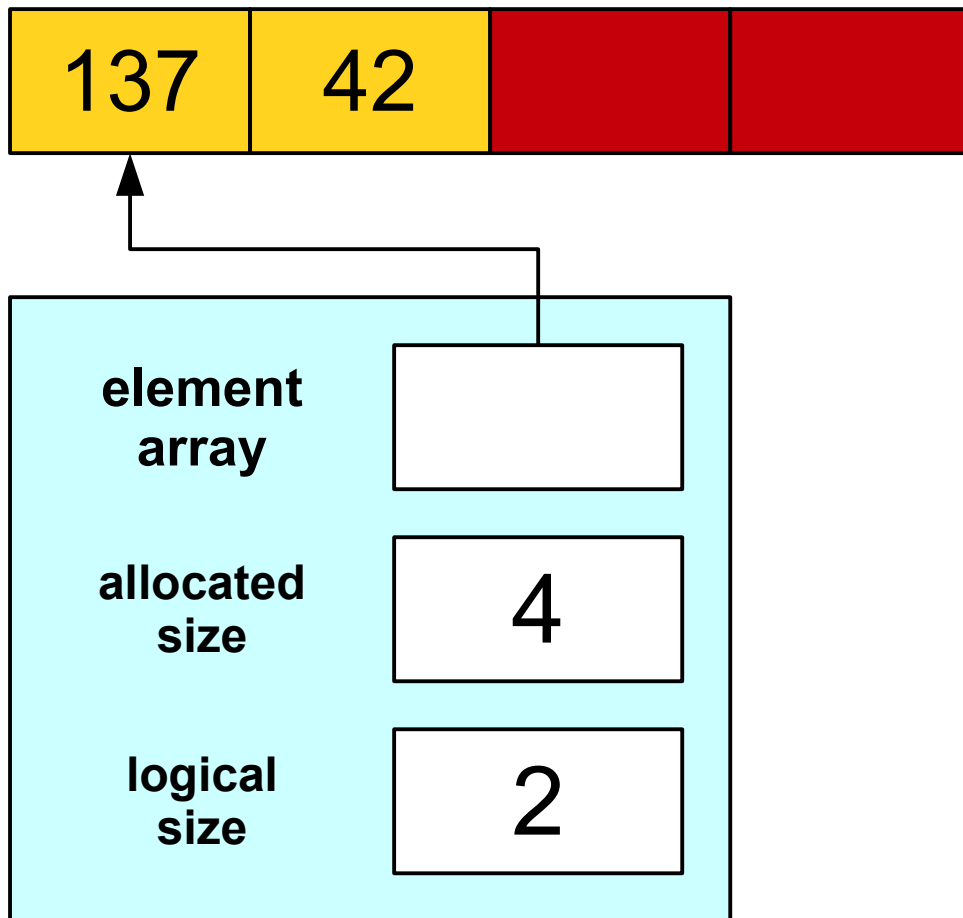
The stack's **allocated size** is the number of slots in the array. Remember - arrays in C++ cannot grow or shrink.

The stack's **logical size** is the number of elements actually stored in the stack. This lets us track how much space we're actually using.

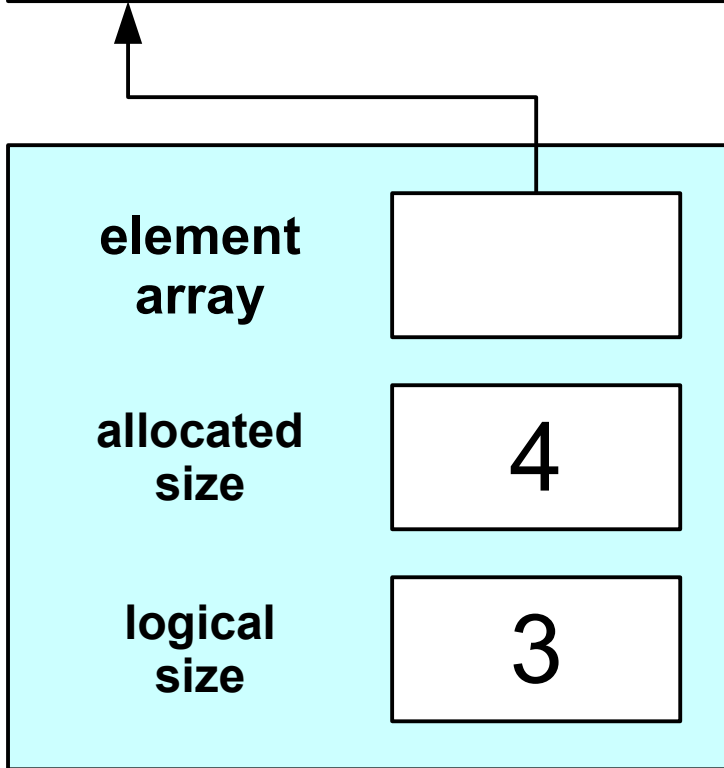
An Initial Idea



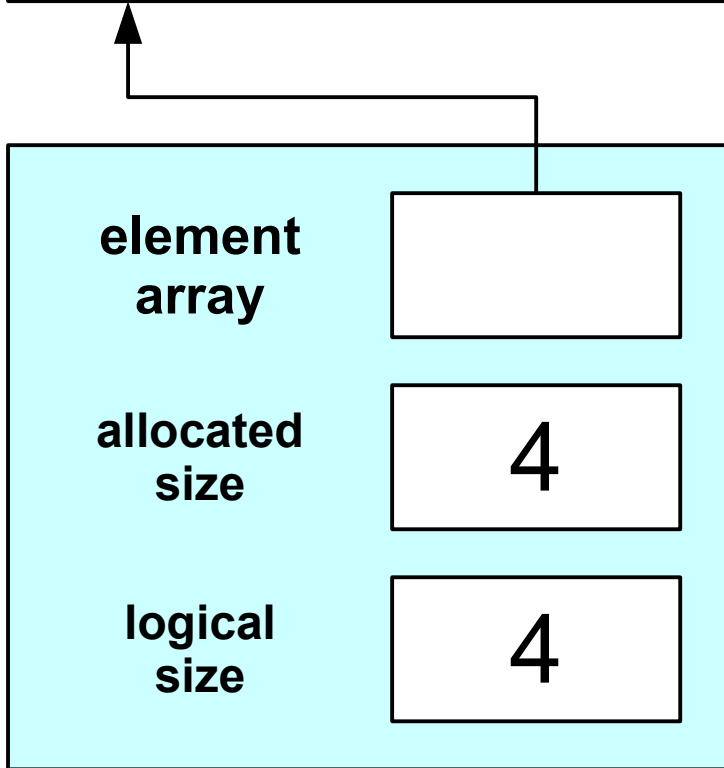
An Initial Idea



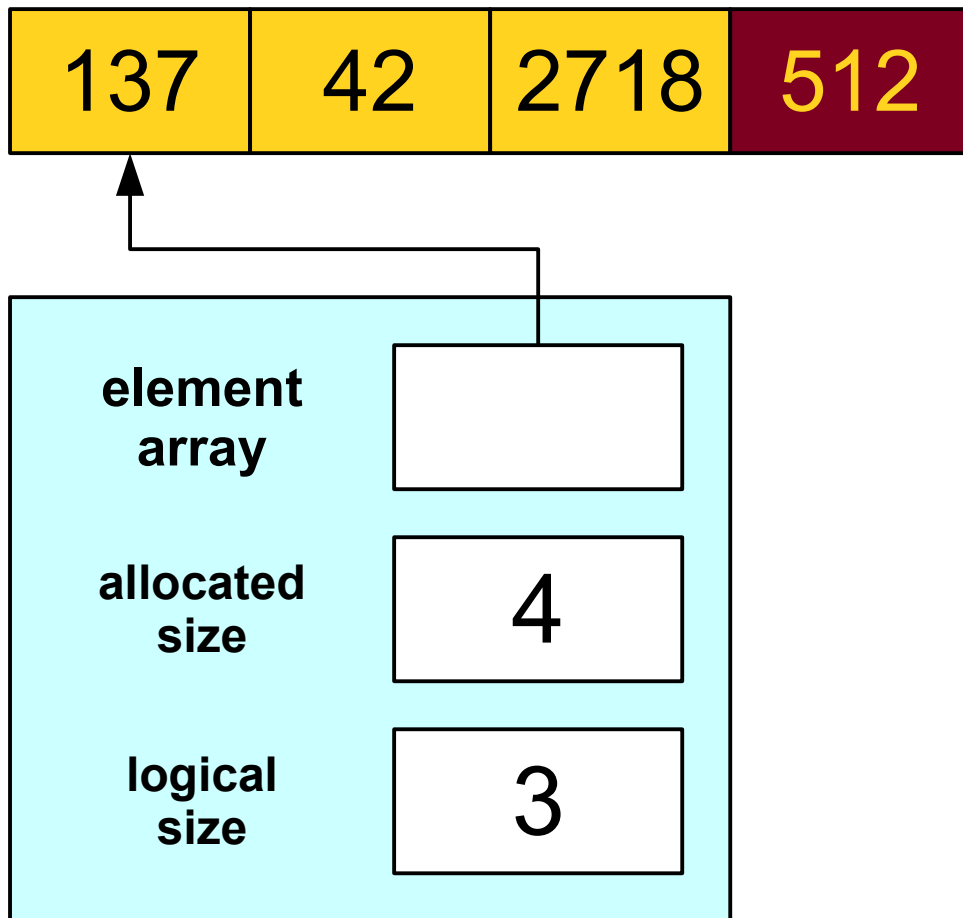
An Initial Idea



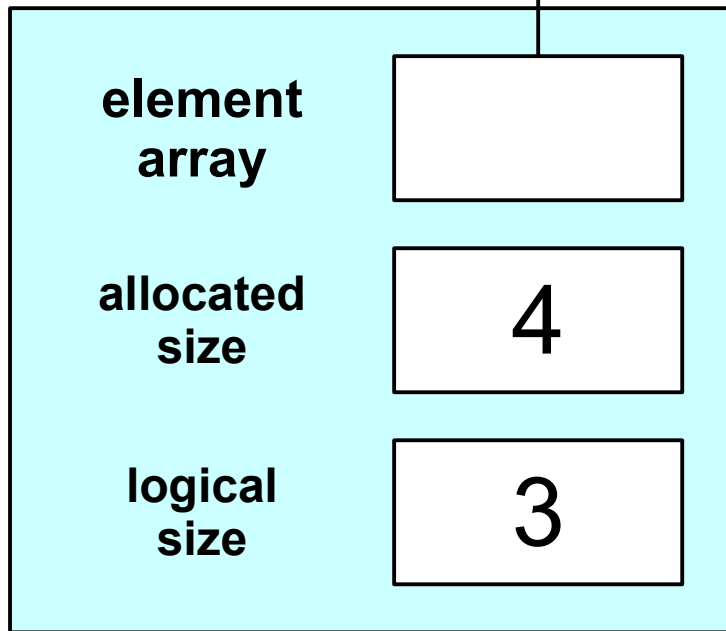
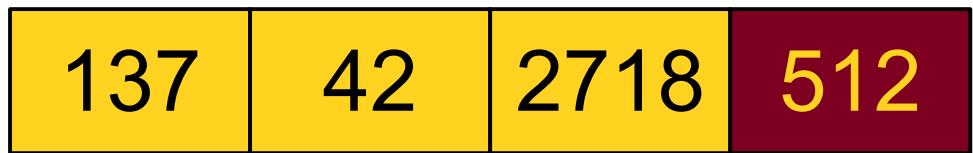
An Initial Idea



An Initial Idea

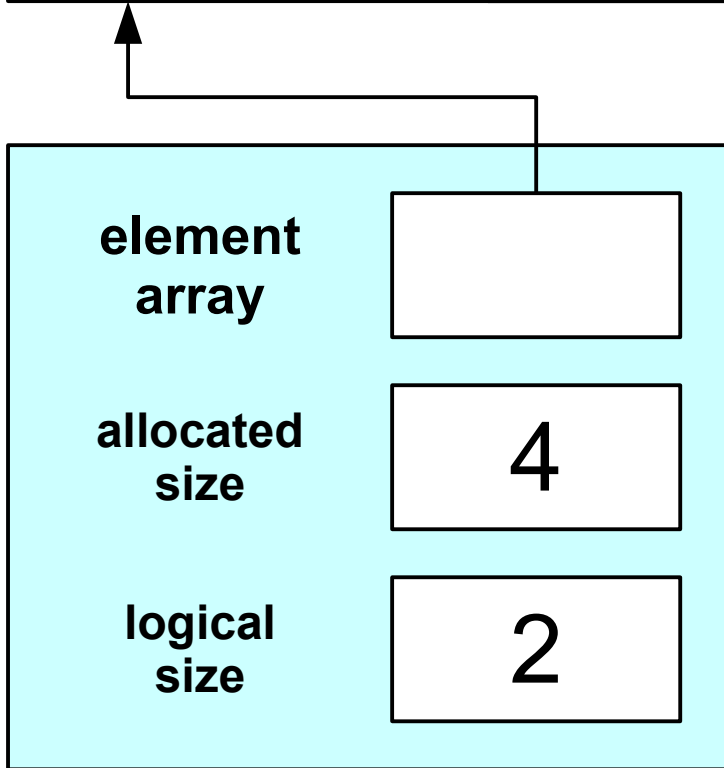


An Initial Idea

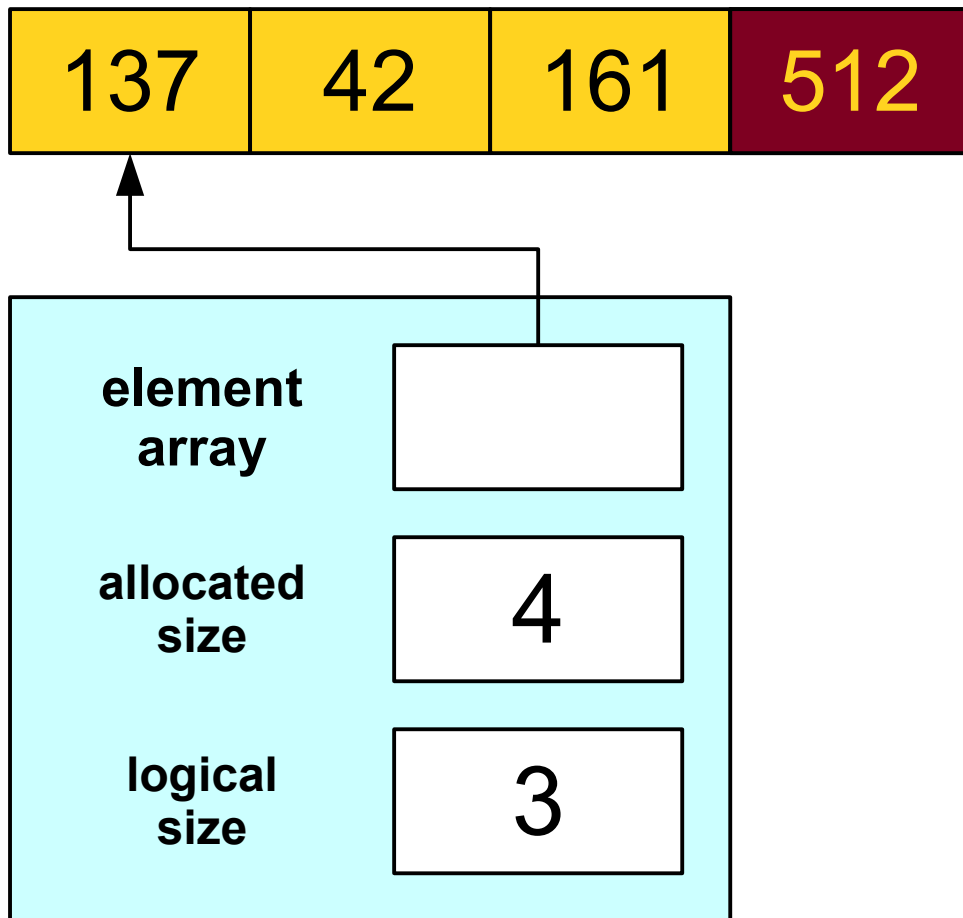


Arrays cannot grow or shrink, so this older value is still technically there in the array. We're just going to pretend it isn't.

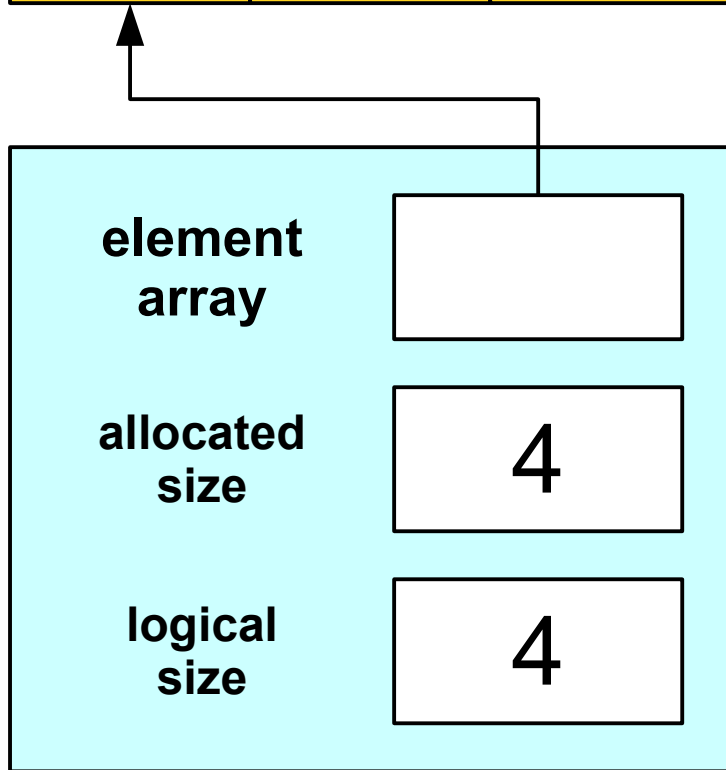
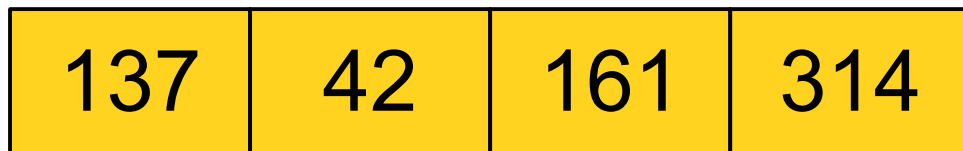
An Initial Idea



An Initial Idea



An Initial Idea



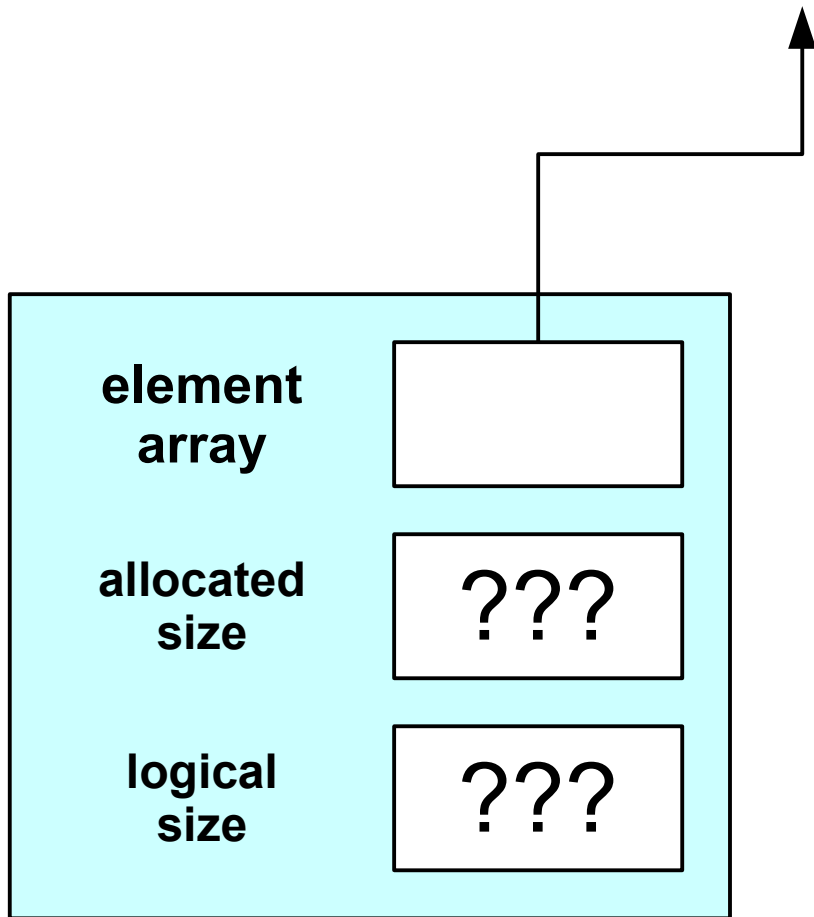
Cradle to Grave

```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave

```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

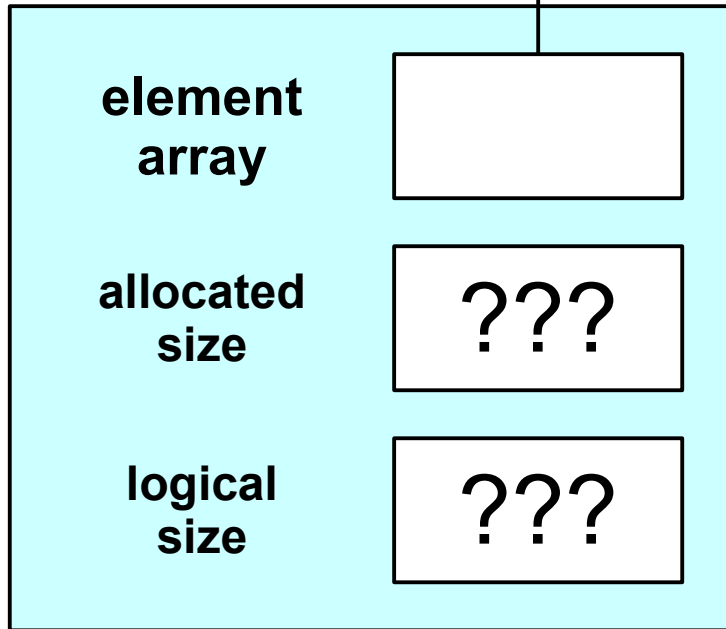

Cradle to Grave



```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave

Undefined behavior!



```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Constructors

- A ***constructor*** is a special member function used to set up the class before it is used.
- The constructor is automatically called when the object is created.
- The constructor for a class named ***ClassName*** has signature

ClassName(args);

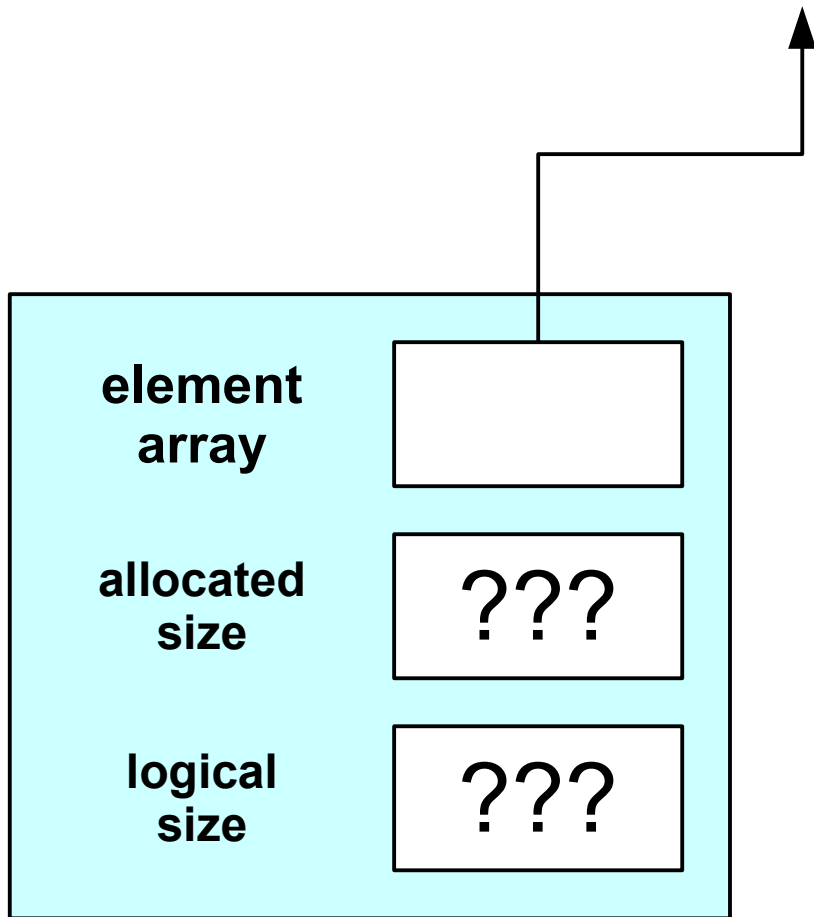
Cradle to Grave, Take II

```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take II

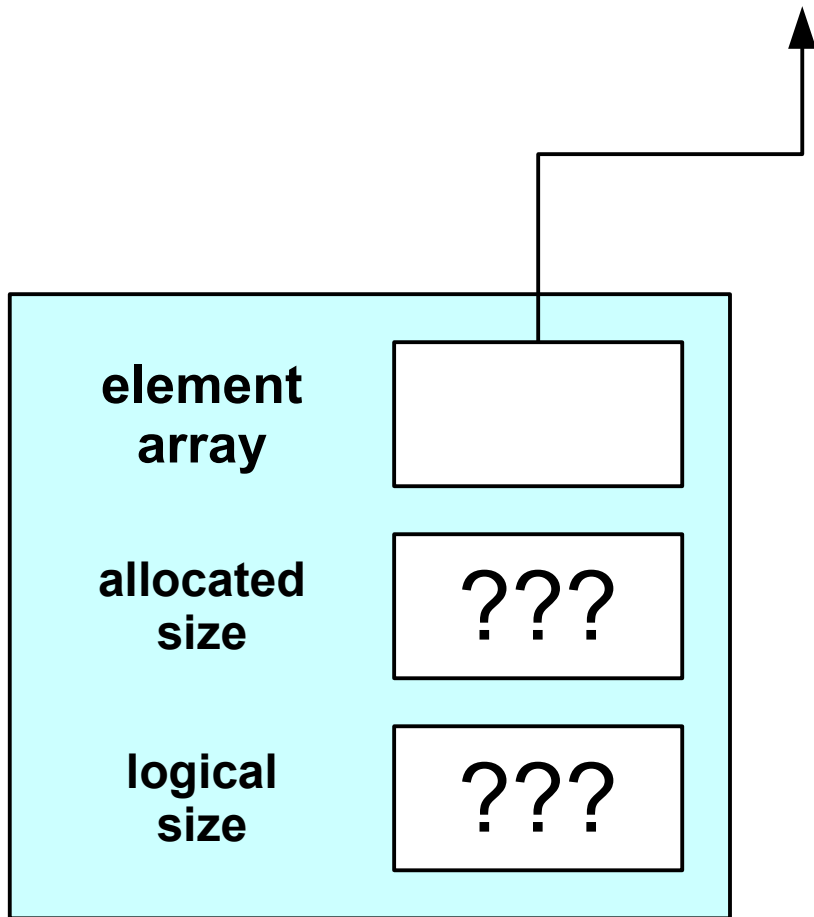
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take II



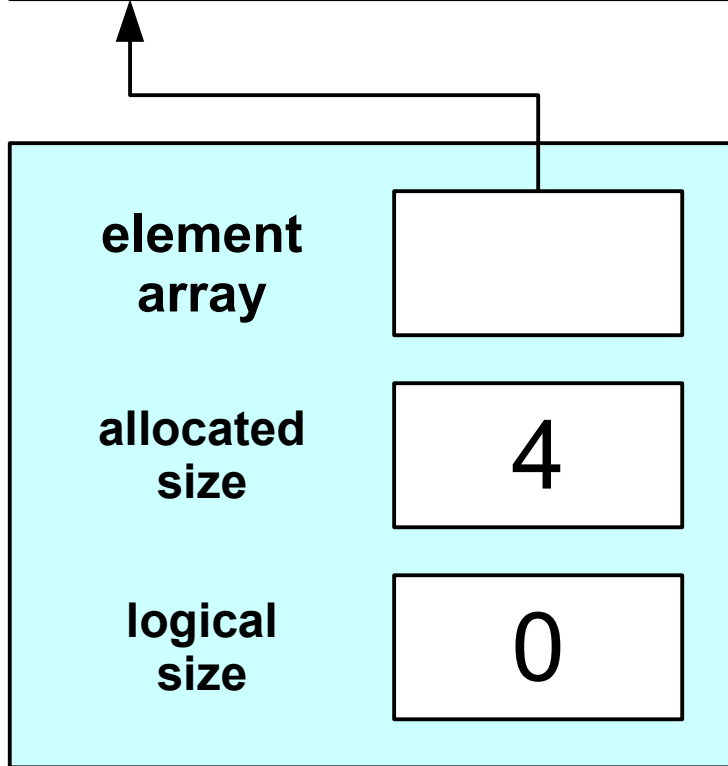
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take II



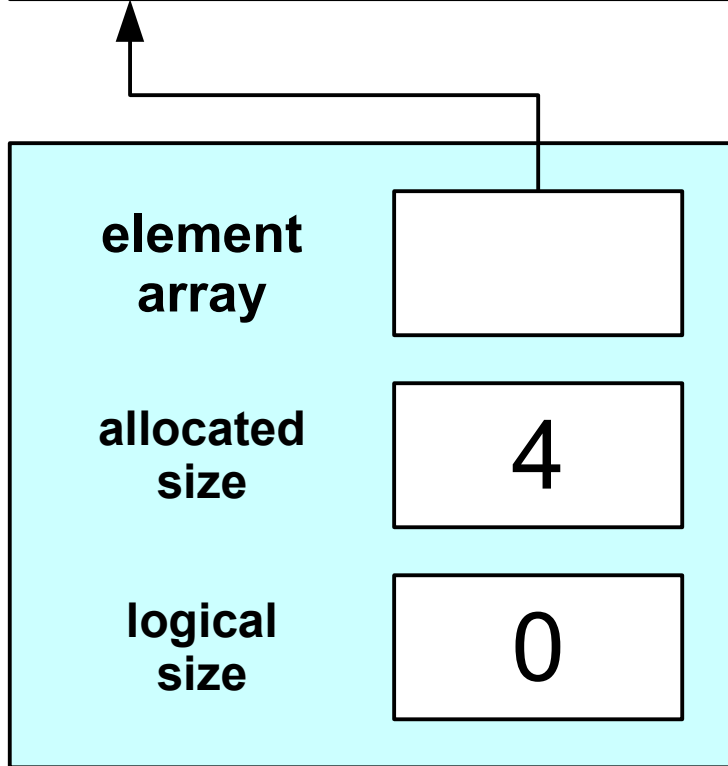
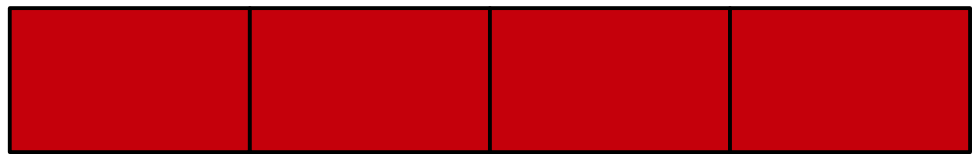
```
int ...  
  
OurStack::OurStack() {  
    logicalSize = 0;  
    allocatedSize = kInitialSize;  
    elems = new int[allocatedSize];  
}  
}
```

Cradle to Grave, Take II



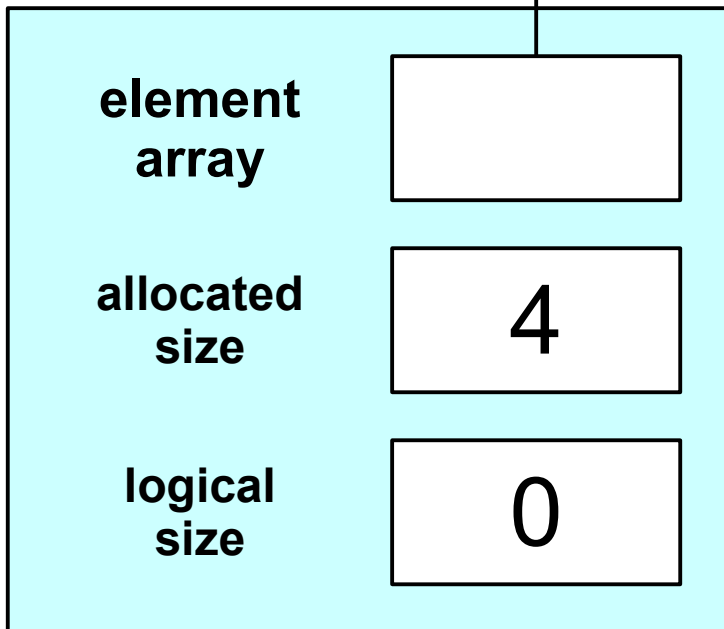
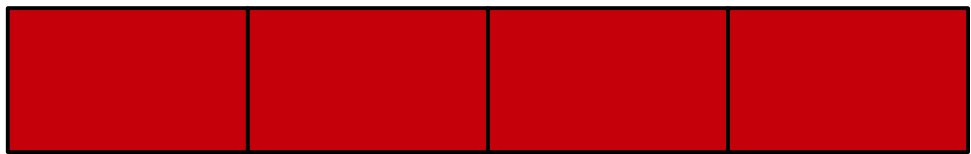
```
int ...  
  
OurStack::OurStack() {  
    logicalSize = 0;  
    allocatedSize = kInitialSize;  
    elems = new int[allocatedSize];  
}  
}
```


Cradle to Grave, Take II



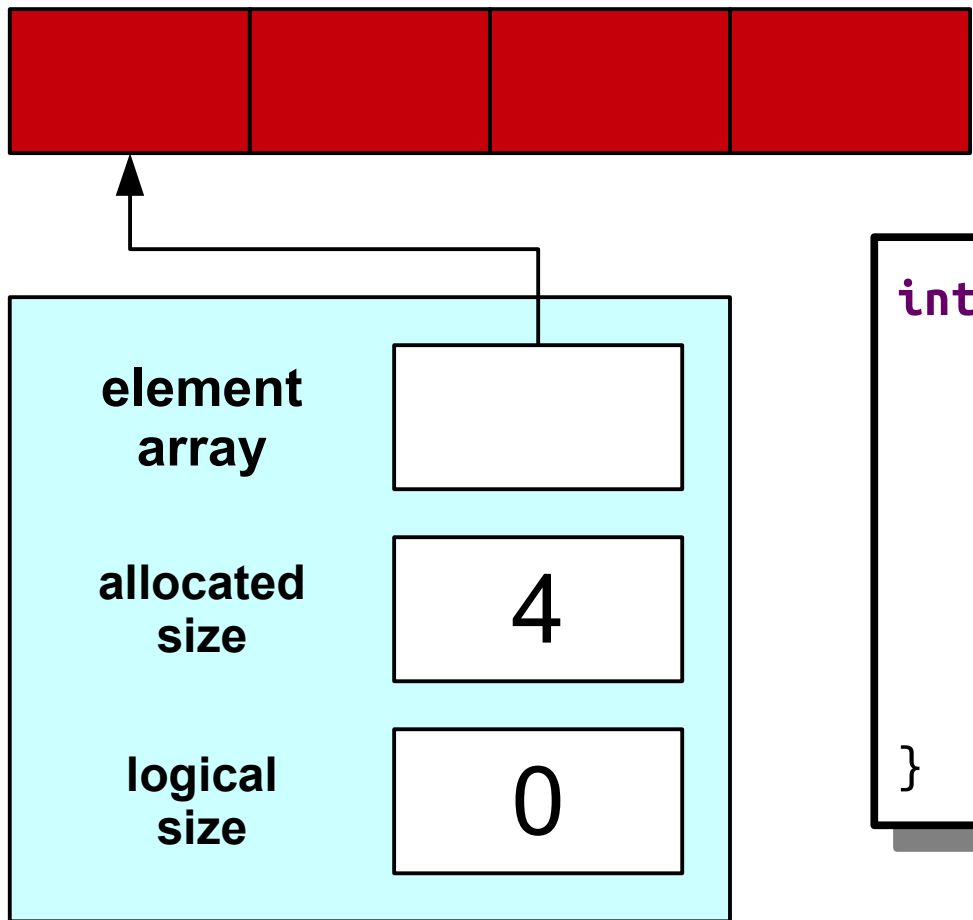
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take II



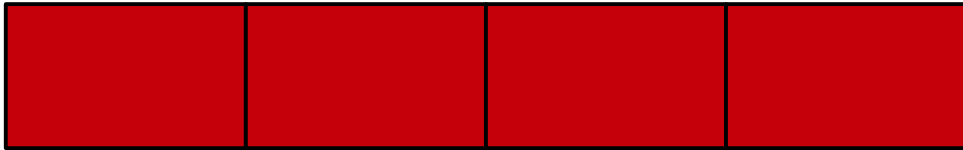
```
int main() {  
    OurStack stack;  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

Cradle to Grave, Take II



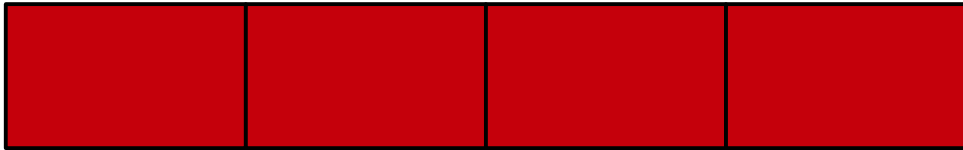
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

Cradle to Grave, Take II



```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

Cradle to Grave, Take II



I am adrift, alone,
condemned to forever
wander meaninglessly.

```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

Destructors

- A ***destructor*** is a special member function responsible for cleaning up an object's memory.
- It's automatically called whenever an object's lifetime ends (for example, if it's a local variable that goes out of scope.)
- The destructor for a class named ***ClassName*** has signature

~ClassName();

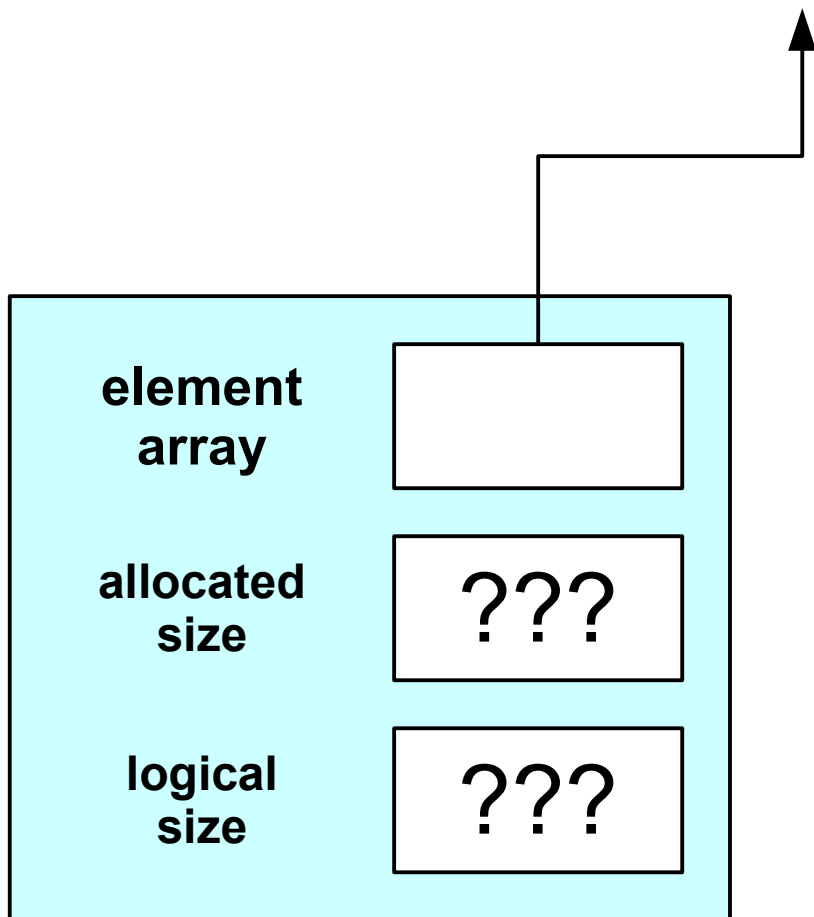
Cradle to Grave, Take III

```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take III

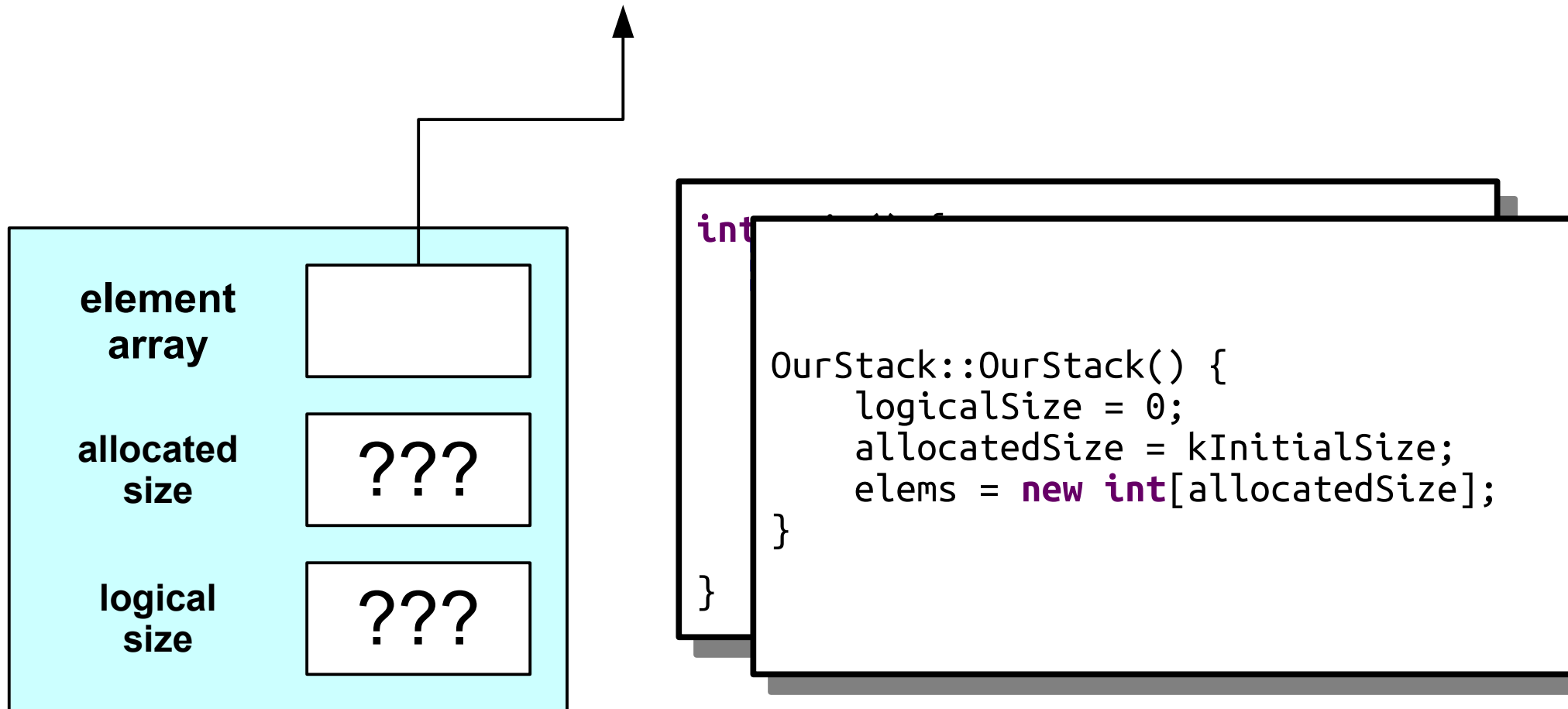
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```


Cradle to Grave, Take III

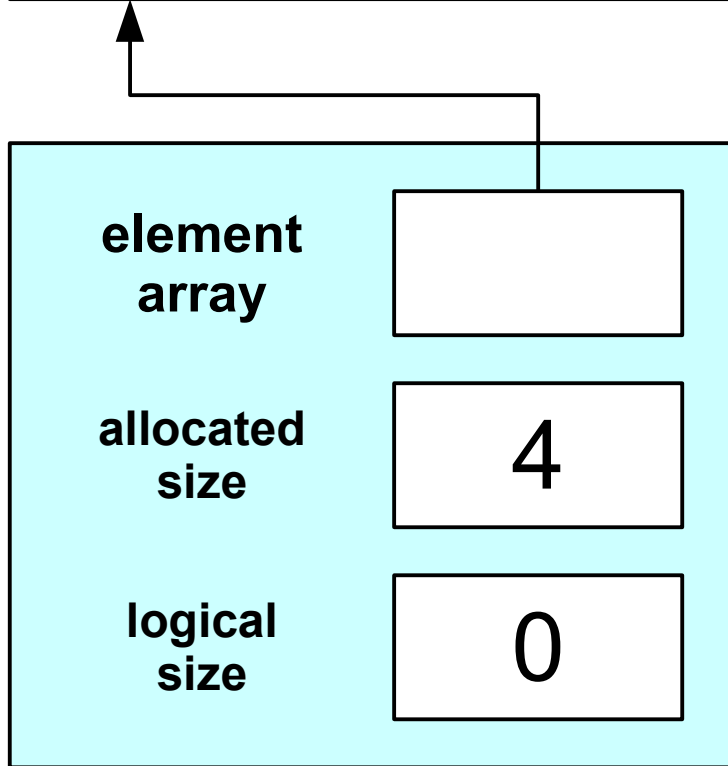
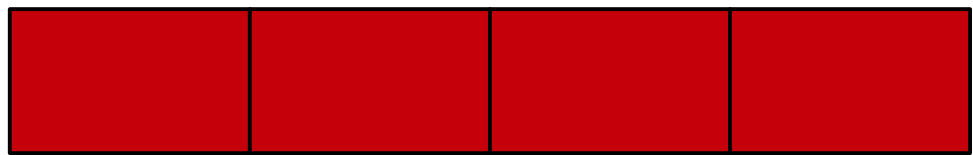


```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take III

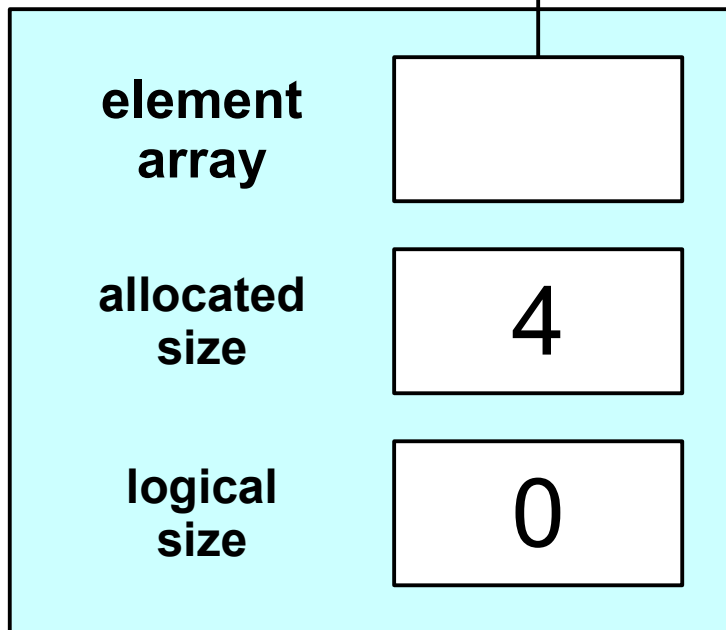
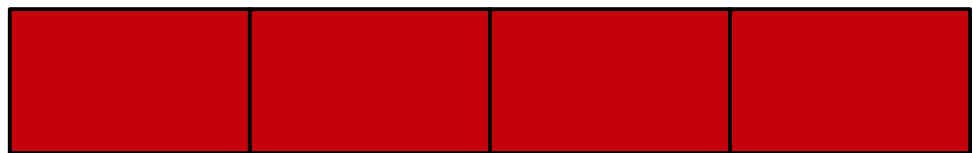


Cradle to Grave, Take III



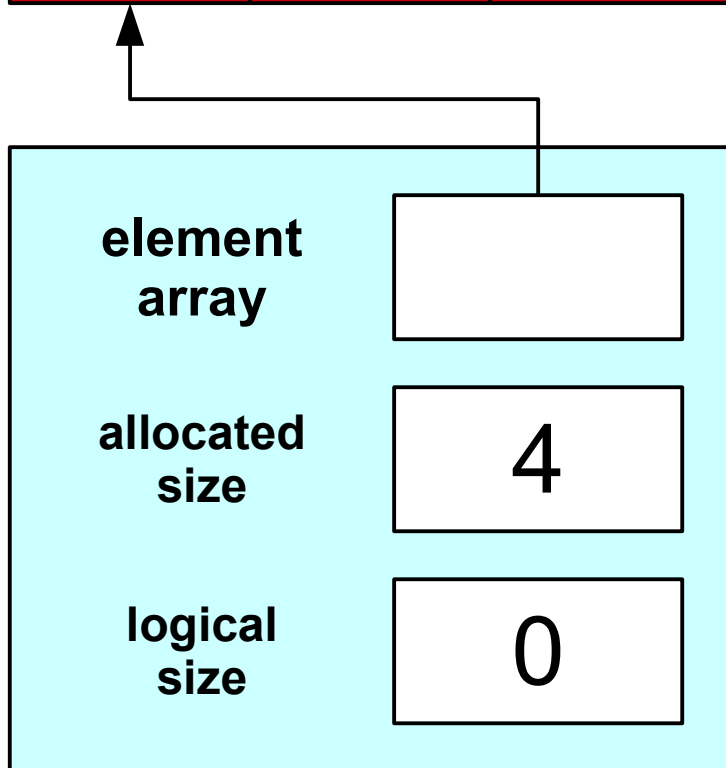
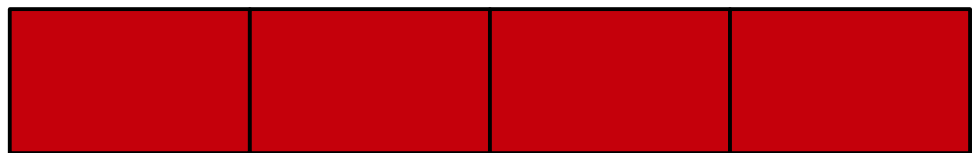
```
int ...  
  
OurStack::OurStack() {  
    logicalSize = 0;  
    allocatedSize = kInitialSize;  
    elems = new int[allocatedSize];  
}  
}
```

Cradle to Grave, Take III



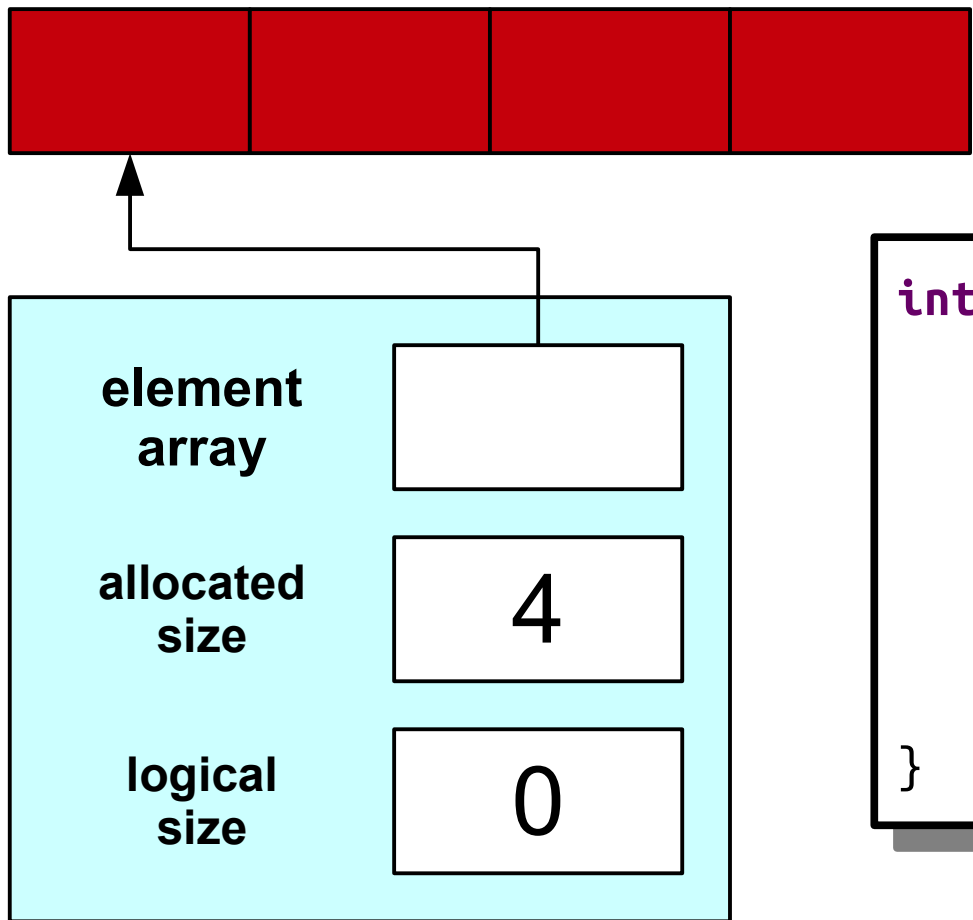
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
  
    return 0;  
}
```

Cradle to Grave, Take III



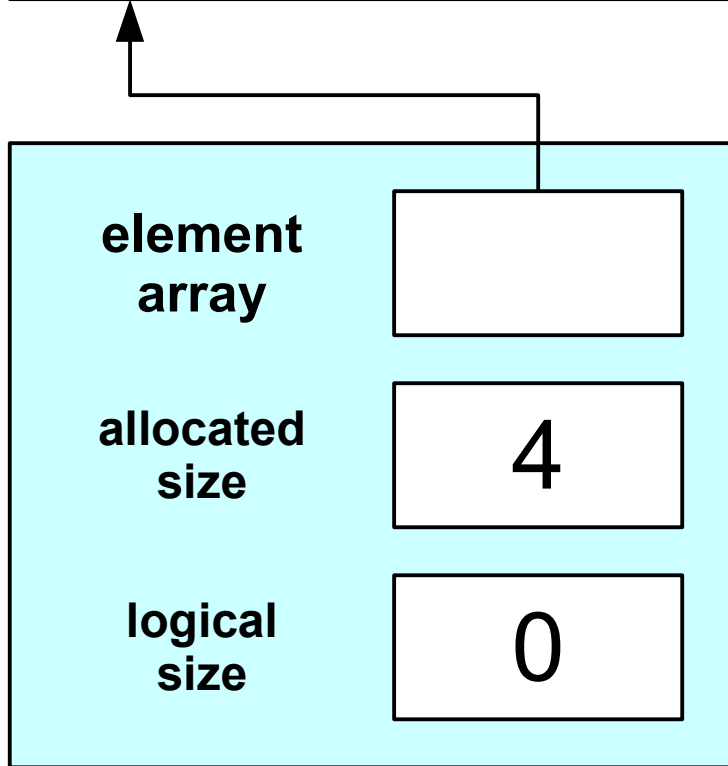
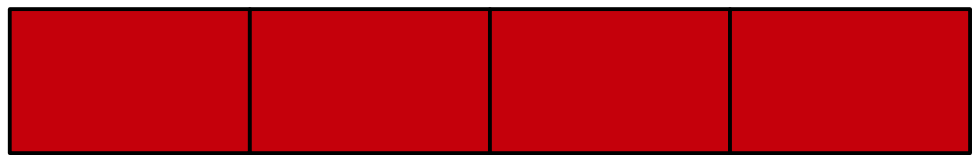
```
int main() {  
    OurStack stack;  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

Cradle to Grave, Take III



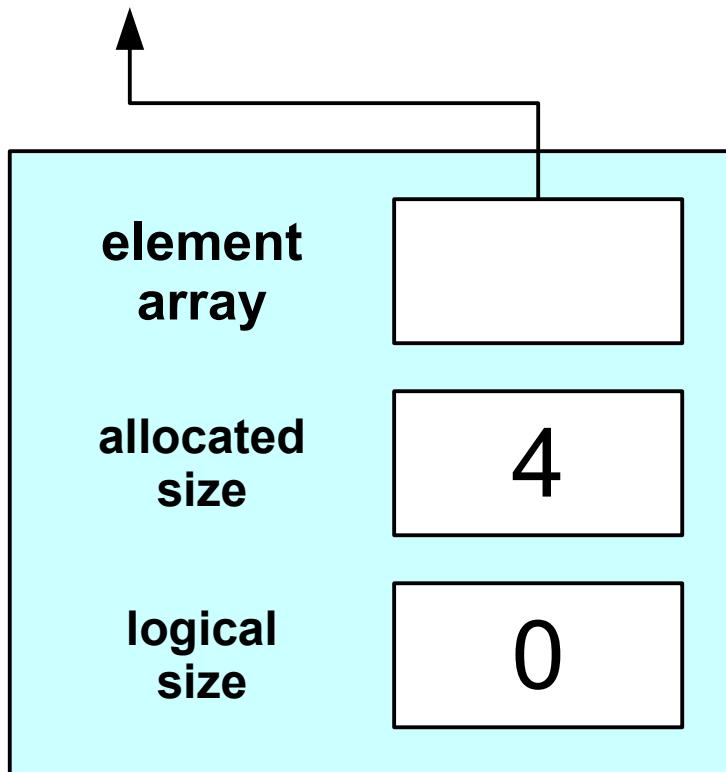
```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

Cradle to Grave, Take III



```
int main() {  
  
    OurStack::~~OurStack() {  
        delete[] elems;  
    }  
}
```

Cradle to Grave, Take III



```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```


Cradle to Grave, Take III

```
int main() {  
    OurStack stack;  
  
    /* The stack lives a rich, happy,  
     * fulfilling life, the kind we  
     * all aspire to.  
     */  
    return 0;  
}
```

To Summarize

- You can create arrays of a fixed size at runtime by using **new**[].
- You are responsible for freeing any memory you explicitly allocate by calling **delete**[].
- Constructors are used to set up a class's internal state so that it's in a good place.
- Destructors are used to free resource that a class allocates.

Next Time

- ***Making Stack Grow!***
 - Different approaches to Stack growth.
 - Analysis of these approaches.
 - The reality: *everything is a tradeoff!*
- ***Implementing the Queue***
 - ... is not too hard when you have a stack!