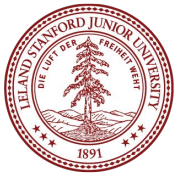
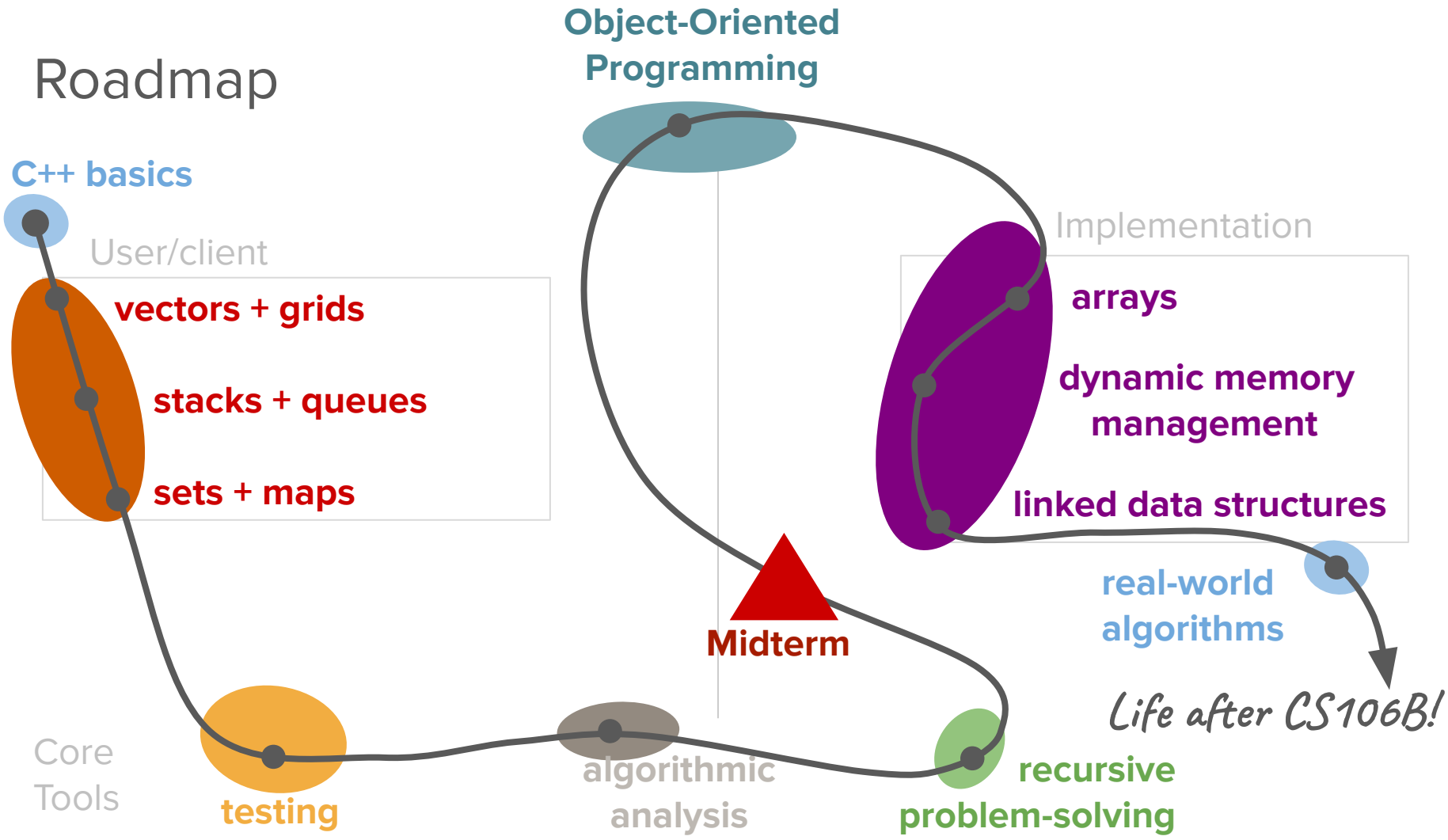
Dynamic Memory and Arrays

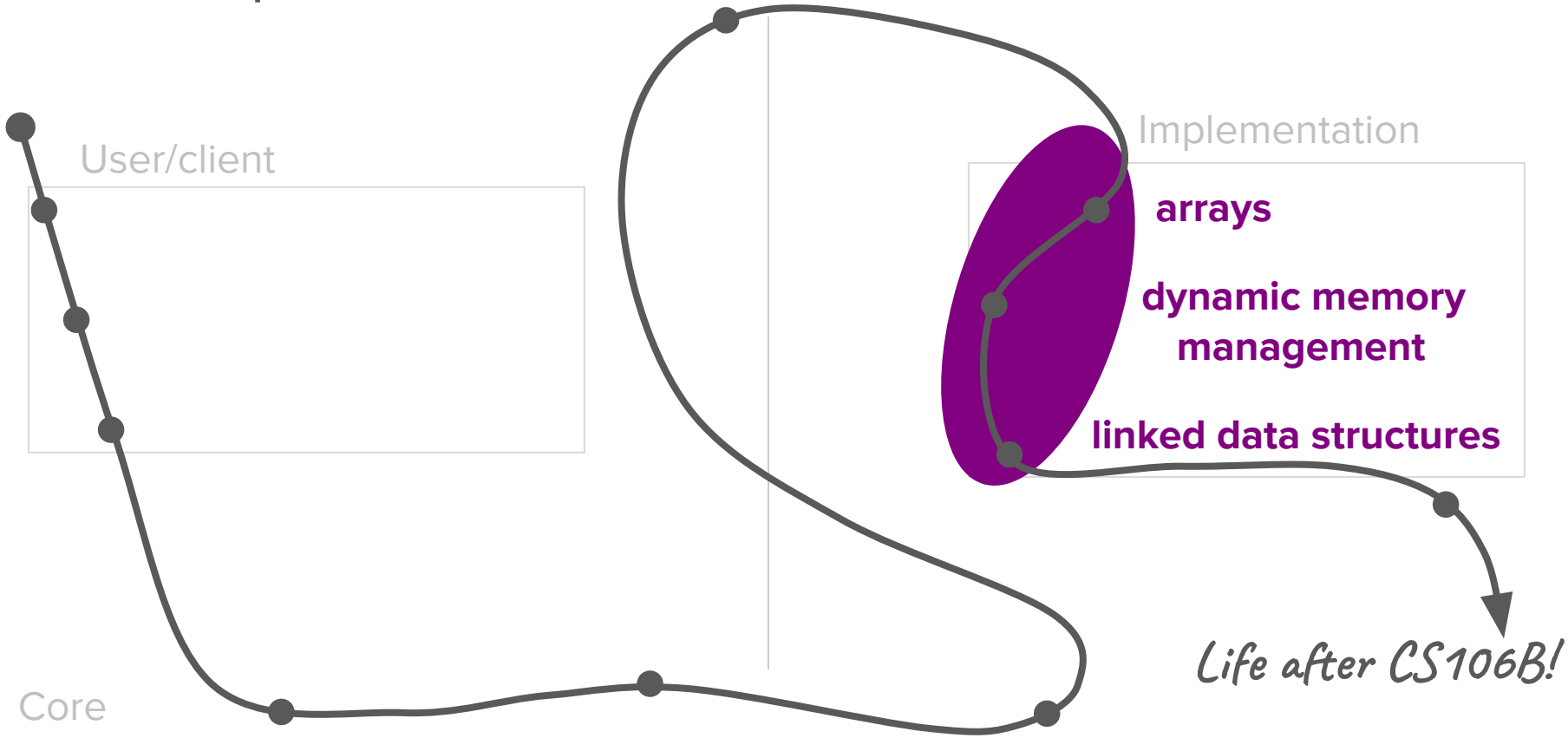
What are real-world examples of classes and abstractions?



Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Midterm
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


User/client
Implementation
arrays
dynamic memory management
linked data structures
Life after CS106B!
Core
Tools

classes

object-oriented programming

abstract data structures
(vectors, maps, etc.)

arrays

dynamic memory
management

linked data structures

testing

algorithmic analysis

recursive problem-solving

classes

object-oriented programming

abstract data structures
(vectors, maps, etc.)

arrays

dynamic memory
management

linked data structures

testing

algorithmic analysis

recursive problem-solving

classes

object-oriented programming

abstract data structures
(vectors, maps, etc.)

arrays

dynamic memory management

linked data structures

testing

algorithmic analysis

recursive problem-solving

classes

object-oriented programming

We've now crossed the abstraction boundary!

abstract data structures
(vectors, maps, etc.)

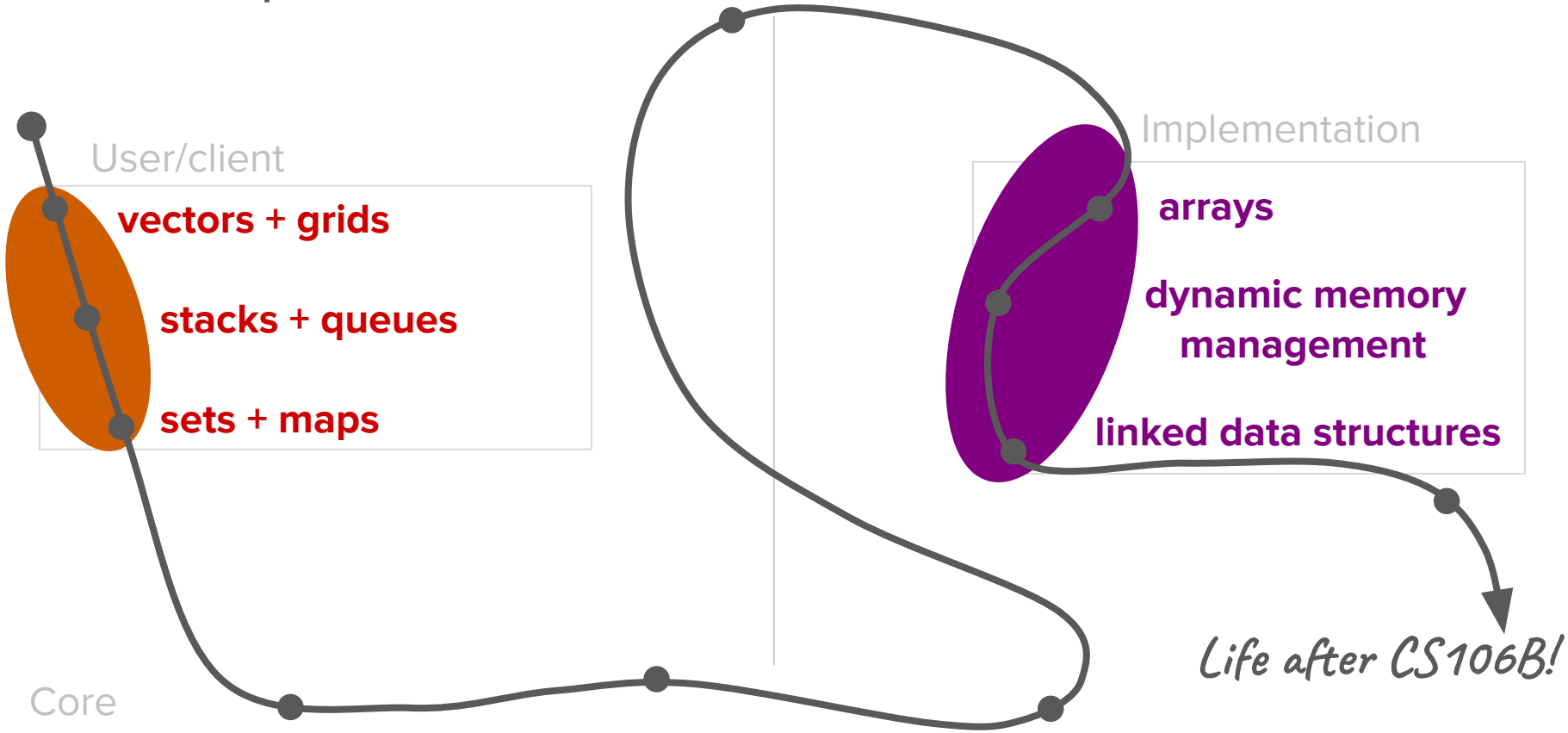
arrays

dynamic memory management

linked data structures

testing *algorithmic analysis* *recursive problem-solving*

Roadmap


User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
Life after CS106B!
Core
Tools

C++ basics

Ordered data

vectors + grids

stacks + queues

sets + maps

Unordered data

Readymade containers are great!

- You can do so much with the ADTs that you have!
  - Write code that sorts names in the U.S. census
  - Use stacks, grids to search for optimal paths in a maze
  - Generate combinations recursively using sets

- You used their interfaces.

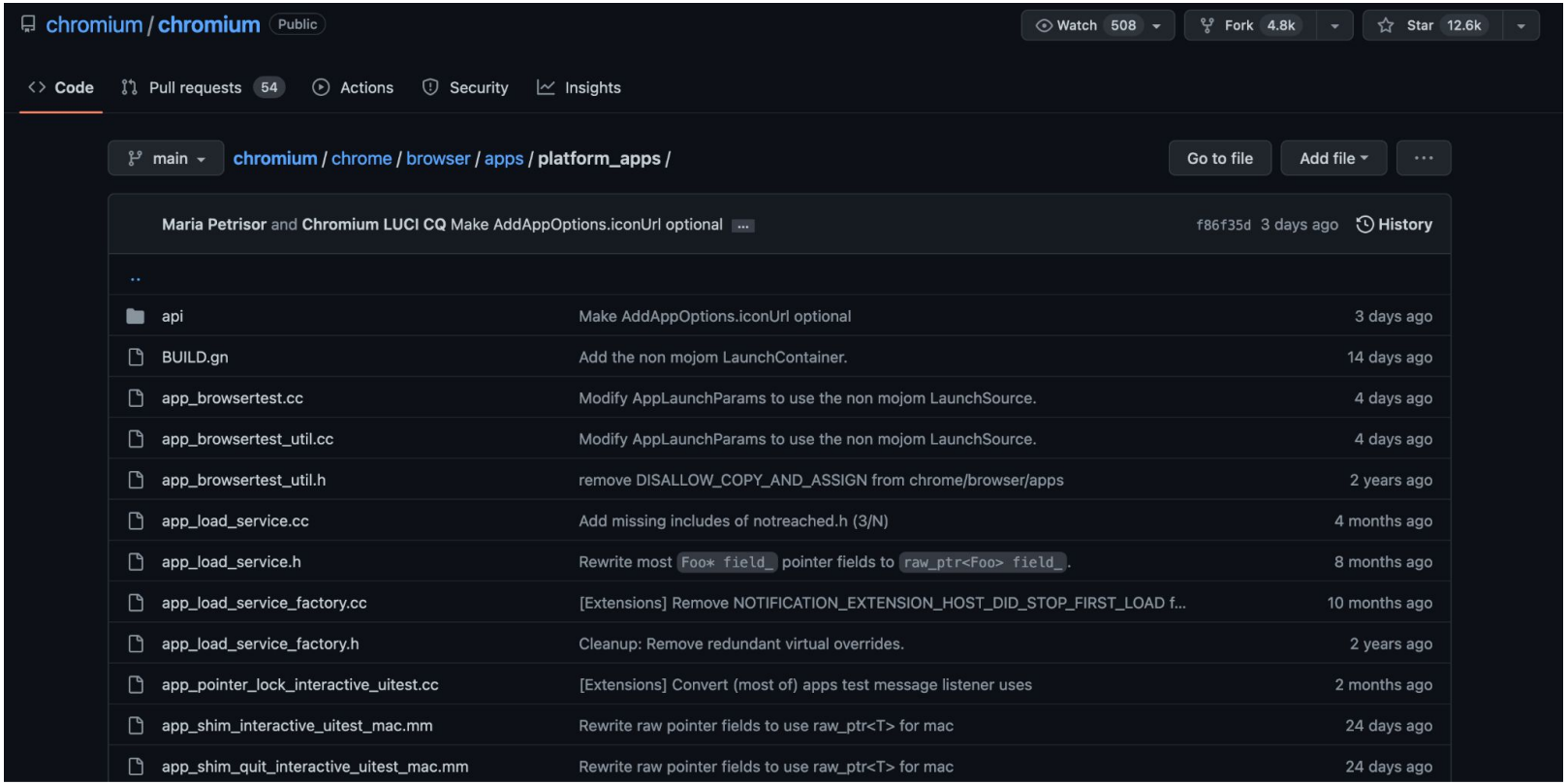
But how are those containers implemented?

- We'll need to learn about more basic building blocks in C++: arrays, pointers.
- Tomorrow, we're building our own vector!

And what if we need custom containers / objects?

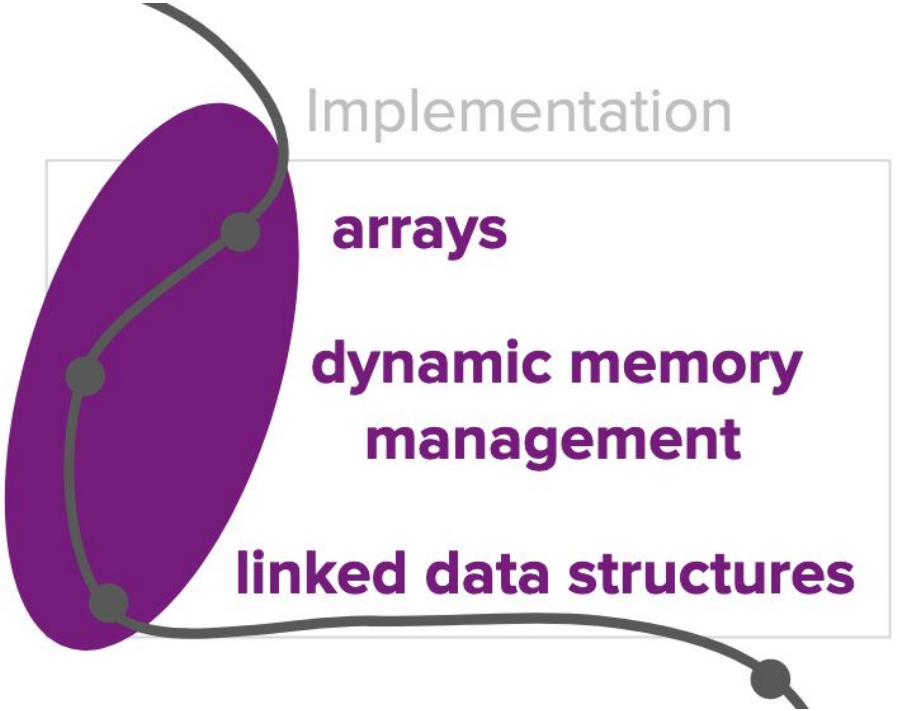
- We have to define our own classes.
- A4, you'll be building a priority queue class!

For example, Google Chrome



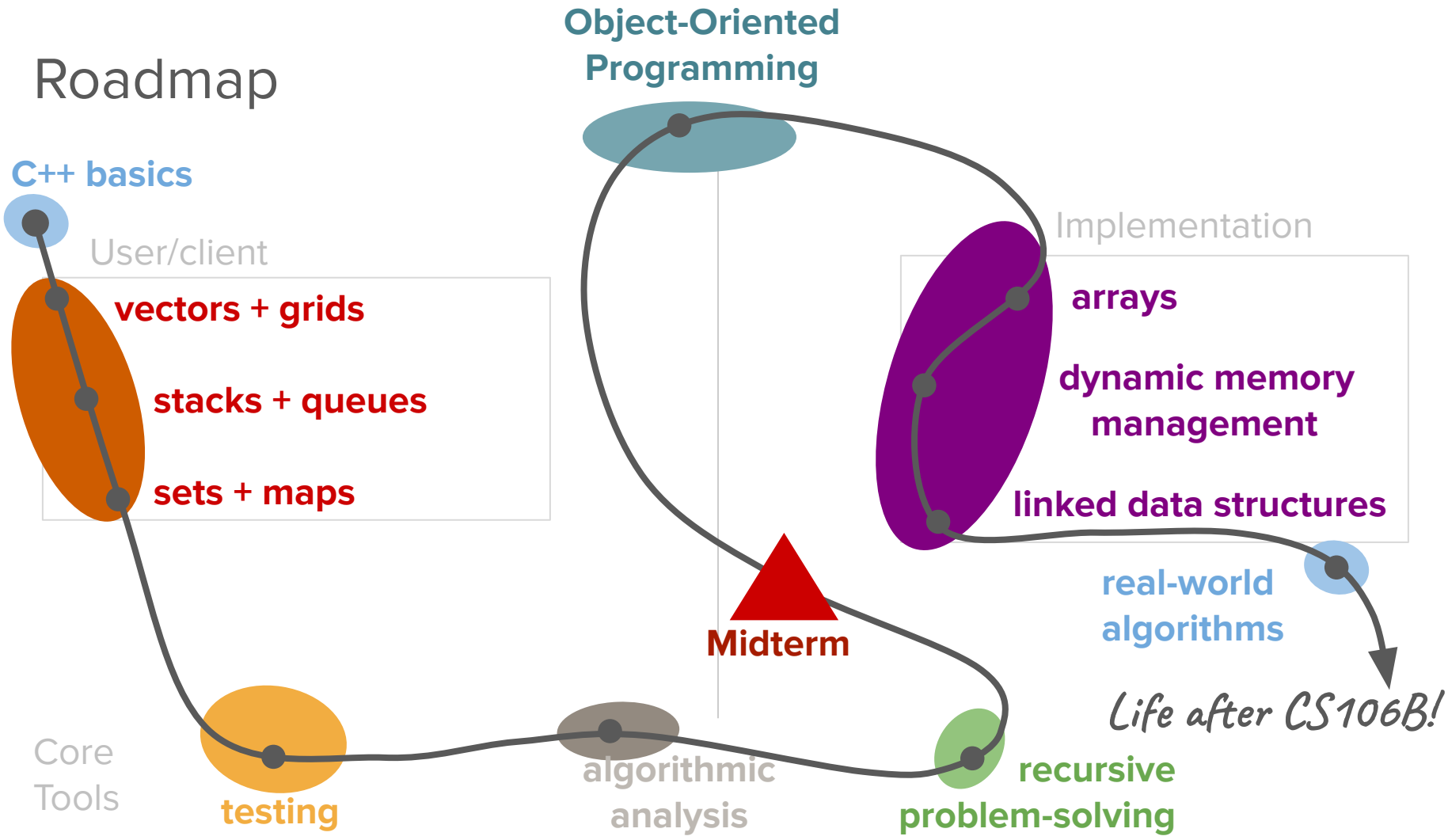
Going under the hood

- We'll need to learn about more basic building blocks in C++.
- We'll need more control of memory management.



Going under the hood

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Midterm
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

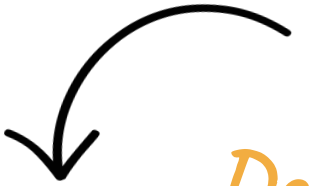
Today's question

What are the fundamental building blocks of data storage provided by C++?

Today's topics

1. Review
2. Dynamic Allocation
3. Arrays
4. Pointers

Review



How do we accomplish this in C++? With classes!

Definition

abstraction

Design that hides the details of how something works while still allowing the user to access complex functionality

Definition

class

A class defines a new data type for our programs to use.

Definition

encapsulation

The process of grouping related information and relevant functions into one unit and defining where that information is accessible

Another way to think about classes...

- A blueprint for a new type of C++ **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

Definition

instance

When we create an object that is our new type, we call this creating an instance of our class.

A class is a type that you define

- Every class has two parts:
  - an **interface** specifying what operations can be performed on instances of the class (this defines the abstraction boundary)
  - an **implementation** specifying how those operations are to be performed

- The only difference between structs + classes are the **encapsulation** defaults.
  - A struct defaults to **public** members (accessible outside the struct itself).
  - A class defaults to **private** members (accessible only inside the class implementation).

Three main parts

- Member variables
  - These are the variables stored within the class
  - Usually not accessible outside the class implementation

- Member functions (methods)
  - Functions you can call on the object
  - E.g. **`vec.add()`**, **`vec.size()`**, **`vec.remove()`**, etc.

- Constructor
  - Gets called when you create the object
  - E.g. **`Vector<int> vec;`**

How do we design a class?

We must specify the 3 parts:

1. Member variables: *What subvariables make up this new variable type?*

2. Member functions: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*

In general, classes are useful in helping us with complex programs where information can be grouped into objects.

Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with **`.h`**) describing what operations the class can perform and what internal state it needs.
  - Create an **implementation file** (typically suffixed with **`.cpp`**) that contains the implementation of the class.

- Clients of the class can then include (using the **`#include`** directive) the header file to use the class.

Struct vs class?

- The only difference between structs + classes are the **encapsulation** defaults.
  - A struct defaults to **public** members (accessible outside the struct itself).
  - A class defaults to **private** members (accessible only inside the class implementation).

"A struct simply *feels* like an open pile of bits with very little in the way of encapsulation or functionality."

"A class *feels* like a living and responsible member of society with intelligent services, a strong encapsulation barrier, and a well defined interface."

CPP Wiki

Structs vs. classes (BankAccount)

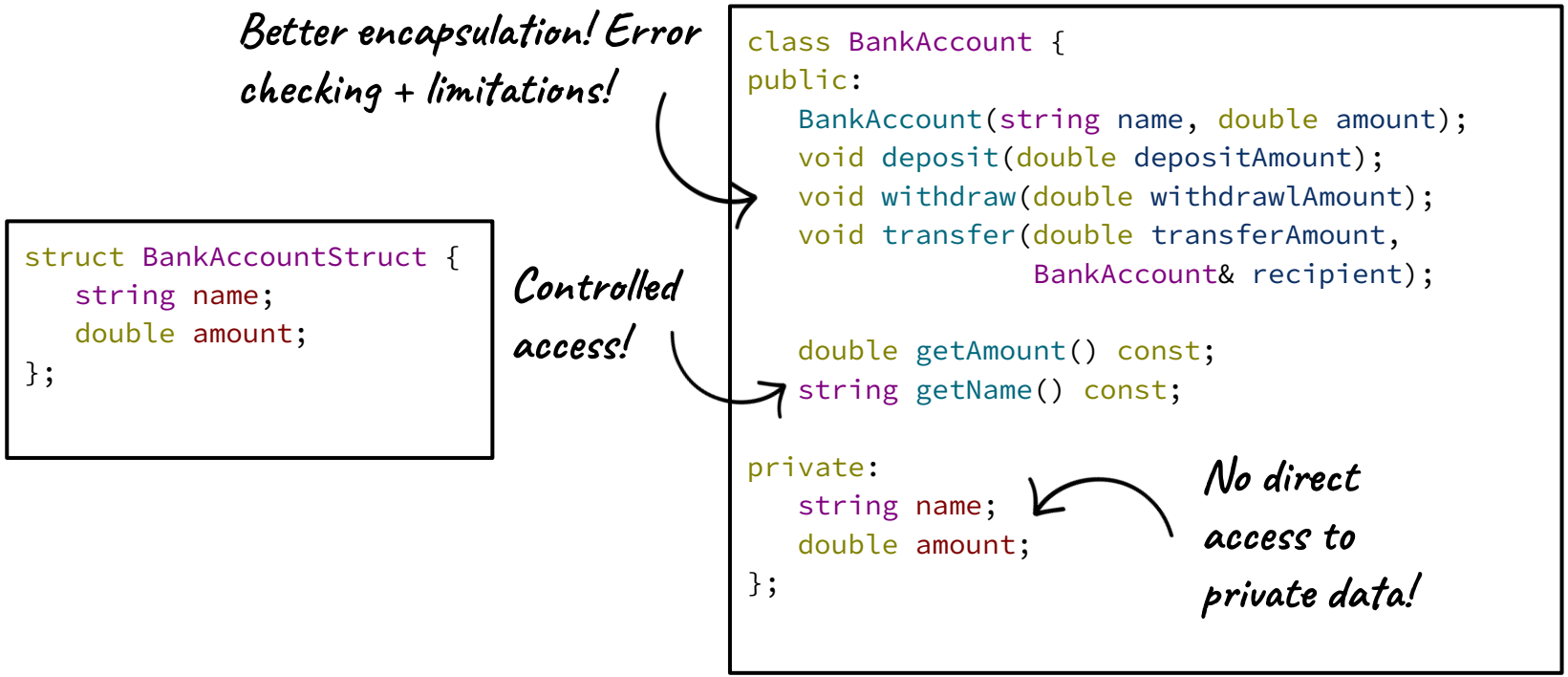
```
struct BankAccountStruct {
    string name;
    double amount;
};
```

```
class BankAccount {
public:
    BankAccount(string name, double amount);
    void deposit(double depositAmount);
    void withdraw(double withdrawlAmount);
    void transfer(double transferAmount,
                  BankAccount& recipient);

    double getAmount() const;
    string getName() const;

private:
    string name;
    double amount;
};
```

Structs vs. classes (BankAccount)



Final Takeaways

- The constructor is a specially defined method for classes that initializes the state of new objects as they are created.
  - Often accepts parameters for the initial state of the fields.
  - Special naming convention defined as **`ClassName()`**
  - You can never directly call a constructor, but one will always be called when declaring a new instance of an object

- **`this`**
  - Refers to the current instance of an object that a method is being called on
  - Similar to the **`self`** keyword in Python and the **`this`** keyword in Java
  - Syntax: **`this->memberVariable`**
  - Common usage: In the constructor, so parameter names can match the names of the object's member variables.

RandomBag Revisited

```
#pragma once
#include "vector.h"

class RandomBag {
public:
  void add(int value);
  int removeRandom();
  int size() const;
  bool isEmpty() const;

private:
  Vector<int> elems;
};
```

```
#pragma once
#include "vector.h"

class RandomBag {
public:
   void add(int value);
   int removeRandom();
   int size() const;
   bool isEmpty() const;

private:
   Vector<int> elems;
};
```

What are the fundamental building blocks of data storage provided by C++?

Getting Storage Space

Getting Storage Space

- The **`Vector`**, **`Stack`**, **`Queue`**, etc. all need storage space to put the elements that they store.

Getting Storage Space

- The **`Vector`**, **`Stack`**, **`Queue`**, etc. all need storage space to put the elements that they store.

- That storage space is acquired using **dynamic memory allocation**.

Getting Storage Space

- The **`Vector`**, **`Stack`**, **`Queue`**, etc. all need storage space to put the elements that they store.

- That storage space is acquired 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.

Arrays

Arrays

- Storage space on computers, which we often refer to as memory, is allocated in organized chunks called **arrays**

Arrays

- Storage space on computers, which we often refer to as memory, is allocated in organized chunks called **arrays**

- An array is a contiguous chunk of space in the computer's memory, split into slots, each of which can contain one piece of information
  - Contiguous means that each slot is located directly next to the others. There are no "gaps".
  - All arrays have a specific type. Their type dictates what information can be held in each slot.
  - Each slot has an "index" by which we can refer to it.

Arrays

- Storage space on computers, which we often refer to as memory, is allocated in organized chunks called **arrays**

- An array is a contiguous chunk of space in the computer's memory, split into slots, each of which can contain one piece of information
  - Contiguous means that each slot is located directly next to the others. There are no "gaps".
  - All arrays have a specific type. Their type dictates what information can be held in each slot.
  - Each slot has an "index" by which we can refer to it.

Arrays

- Storage space on computers, which we often refer to as memory, is allocated in organized chunks called **arrays**

- An array is a contiguous chunk of space in the computer's memory, split into slots, each of which can contain one piece of information
  - Contiguous means that each slot is located directly next to the others. There are no "gaps".
  - All arrays have a specific type. Their type dictates what information can be held in each slot.
  - Each slot has an "index" by which we can refer to it.

Index:	0	1	2	3	4	5	6

Dynamically Allocating Arrays

Dynamically Allocating Arrays

- First, declare a variable that will point at the newly-allocated array. If the array elements have type `Type`, the pointer will have type `Type*`.
  - e.g. `int*, string*, Vector<double>*`

Dynamically Allocating Arrays

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

Dynamically Allocating Arrays

- First, declare a variable that will point at the newly-allocated array. If the array elements have type **`Type`**, the pointer will have type **`Type*`**.
  - 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:

```
Type* arr;
arr = new Type[size];
```

Dynamically Allocating Arrays

- Declare a variable that will point at the newly-allocated array. If the array elements have type **`Type`**, the pointer will have type **`Type*`**.
  - 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:

```
Type* arr;
arr = new Type[size];
```

- Or, in the same line:

```
Type* arr = new Type[size];
```

Pointers

Pointers

- A pointer is a brand new data type that becomes very prominent when working with dynamically allocated memory.

Pointers

- A pointer is a brand new data type that becomes very prominent when working with dynamically allocated memory.

- Just like all other data types, pointers take up space in memory and can store specific values.

Pointers

- A pointer is a brand new data type that becomes very prominent when working with dynamically allocated memory.

- Just like all other data types, pointers take up space in memory and can store specific values.

- The meaning of these values is what's important. **A pointer always stores a memory address**, which is like the specific coordinates of where a piece of memory exists on the computer.

Pointers

- A pointer is a brand new data type that becomes very prominent when working with dynamically allocated memory.

- Just like all other data types, pointers take up space in memory and can store specific values.

- The meaning of these values is what's important. **A pointer always stores a memory address**, which is like the specific coordinates of where a piece of memory exists on the computer.

- Thus, they quite literally "point" to another location on your computer.

Announcements

Announcements

- Midterm grades
  - Grades will be released shortly after class today via Gradescope (should receive email)
  - We want you to go through your feedback and reflect on your learning/mastery!
  - To encourage this, your section leaders will be offering mid-quarter check-in meetings
    - Meet with your SL and discuss your midterm performance, your thoughts on your mastery of the content from the first 5 weeks, your plans for the rest of the quarter, etc.
    - **If you attend AND engage in thoughtful discussion you earn back ⅓ the missed points.**
    - To participate: submit a **brief reflection (2-3 sentences is fine) on areas you want to focus on** to the "Midterm Check-In" assignment on Paperless. Then use the IG Scheduling feature to sign up for time slot with your SL.
- Assignment 3 is due **Tuesday, July 19 at 11:59pm** with a **24-hour grace.**
- **Final Project Proposal** due **Sunday, July 24 at 11:59 pm.**
- Weekly announcements will be posted tonight.

Dynamic Allocation Example

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

numValues

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

numValues

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

numValues

arr

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

numValues

arr

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

numValues

arr

Index: 0 1 2 3 4 5 6

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

numValues

arr

Index: 0 1 2 3 4

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

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

numValues

arr

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

Index: 0 1 2 3 4 5 6

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

numValues

arr

i

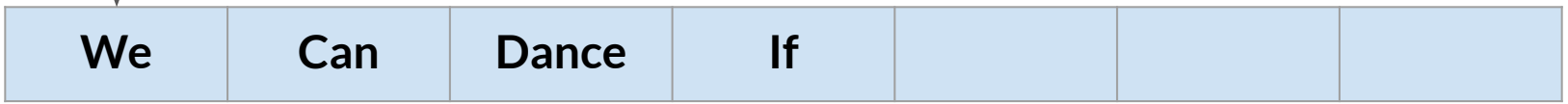
Index: 0 1 2 3 4 5 6

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

numValues

arr

i



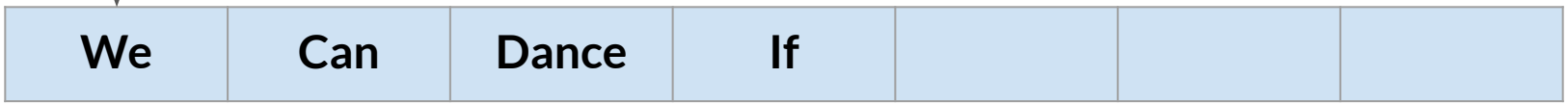
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



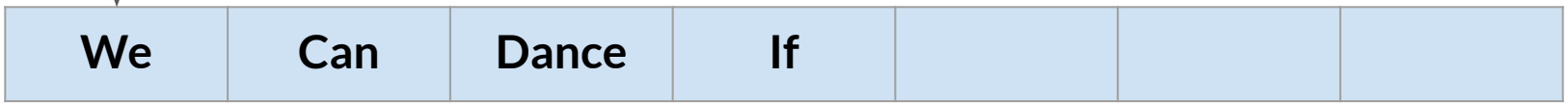
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



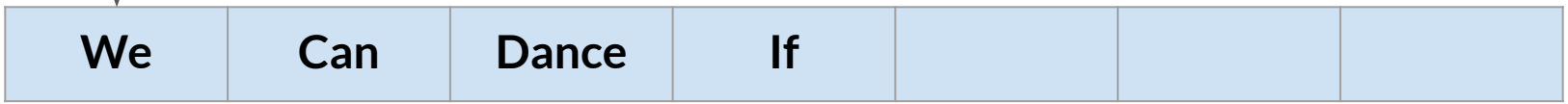
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



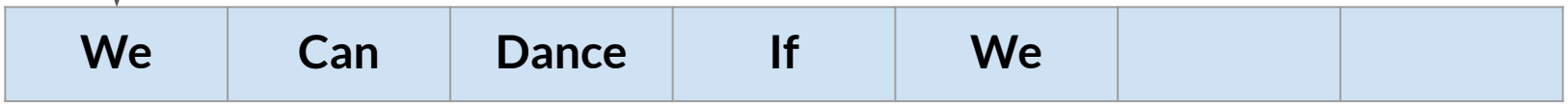
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



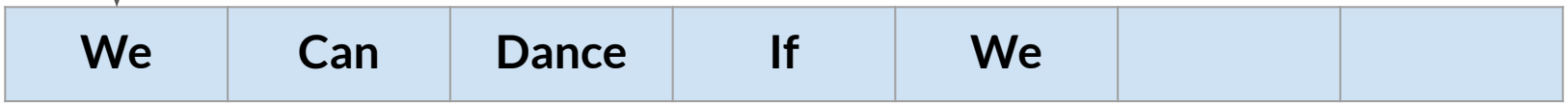
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



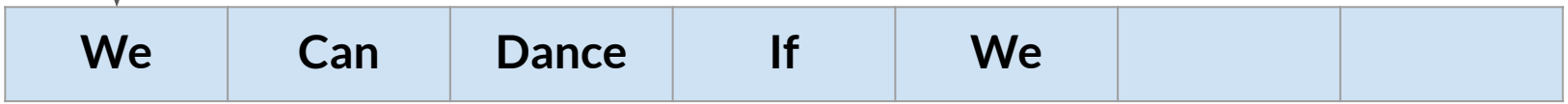
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



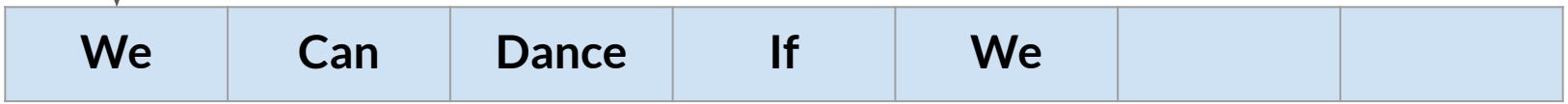
Index:	0	1	2	3	4	5	6

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

numValues

arr

i



Index:	0	1	2	3	4	5	6

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

numValues

arr

i

We	Can	Dance	If	We	Want	
0	1	2	3	4	5	6

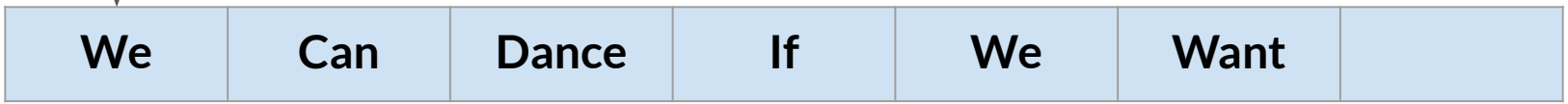
Index:

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

numValues

arr

i



Index:	0	1	2	3	4	5	6

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

numValues

arr

i

We	Can	Dance	If	We	Want	
0	1	2	3	4	5	6

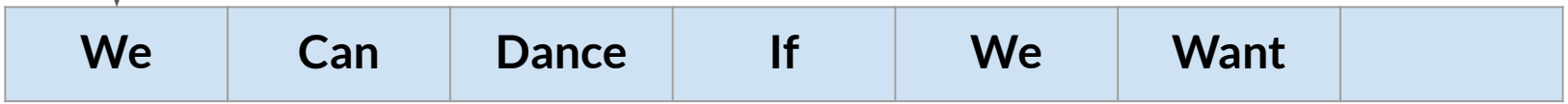
Index:

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

numValues

arr

i



Index:	0	1	2	3	4	5	6

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

numValues

arr

i

We	Can	Dance	If	We	Want	To
0	1	2	3	4	5	6

Index:

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

numValues

arr

i

We	Can	Dance	If	We	Want	To
0	1	2	3	4	5	6

Index:

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

numValues

arr

i

We	Can	Dance	If	We	Want	To
0	1	2	3	4	5	6

Index:

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

numValues

arr

i

We	Can	Dance	If	We	Want	To
0	1	2	3	4	5	6

Index:

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

```
0: We
1: Can
2: Dance
3: If
4: We
5: Want
6: To
```

numValues

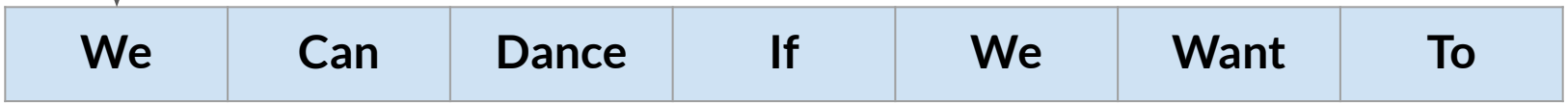
arr

We	Can	Dance	If	We	Want	To
Index: 0	1	2	3	4	5	6

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

numValues

arr



Index:	0	1	2	3	4	5	6

Arrays

- Arrays are allocated with a **fixed size that you can't subsequently change.**
- Even though arrays have a fixed size, **C++ does not make that size available to the programmer.**
    - As a result, programs that work with arrays typically need an additional variable to keep track of the number of elements.
- When we use arrays to build classes, you use pointers and `new` to allocate and keep track of the memory
- C++ performs **no built-in bounds-checking t**o ensure that the elements you select are actually present in the array.

Attendance ticket:

https://tinyurl.com/whylearnarray

Please don’t send this link to students who are not here. It’s on your honor!

Pitfalls and Dangers

Pitfalls and Dangers

Pitfalls and Dangers

- C++’s language philosophy prioritizes speed over safety and simplicity.

Pitfalls and Dangers

- 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.”

Pitfalls and Dangers

- 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*.

Pitfalls and Dangers

- C++'
- The
  - shrir
- The

licity.

ff the

What are potential examples of "undefined behavior" that could occur if you access beyond the bounds of an array? (select all that apply)

- Nothing happens
- You get a random (garbage) value back
- Your program crashes
- You make your computer vulnerable to a hacker takeover
- You make the front page of the New York Times

A brief interlude for some ethics + real world consequences...

"All the News That's Fit to Print"

The New York Times

Late Edition
New York: Today, partly sunny, milder. High 59-64. Tonight, mostly cloudy. Low 48-54. Tomorrow, cloudy, windy, rain developing. High 57-62. Yesterday: High 58, low 41. Details, page D16.

VOL. CXXXVIII... No. 47,679 NEW YORK, FRIDAY, NOVEMBER 4, 1988 35 CENTS

The New York Times/Jon Wilson

Gov. Michael S. Dukakis having his picture taken by a 10-year-old fan at a town meeting in Fairless Hills, Pa., during a tour of the Northeast in which he emphasized the drug problem. Page A19. Vice President Bush addressed supporters a rally in Columbus, Ohio. Less than a week after Mr. Dukakis acknowledged being a liberal, Mr. Bush said yesterday that "this election is not about labels." Page A18.

'Virus' in Military Computers Disrupts Systems Nationwide

By JOHN MARKOFF

In an intrusion that raises questions about the vulnerability of the nation's computers, a Department of Defense network has been disrupted since Wednesday by a rapidly-spreading "virus" program apparently introduced by a computer science student.

The program reproduced itself through the computer network, making hundreds of copies in each machine it reached, effectively clogging systems linking thousands of military, corporate and university computers around the nation and preventing them from doing additional work. The virus is thought not to have destroyed any files.

By late yesterday afternoon computer experts were calling the virus the largest assault ever on the nation's computers.

'The Big Issue'

"The big issue is that a relatively benign software program can virtually bring our computing community to its knees and keep it there for some time," said Chuck Cole, deputy computer security manager at Lawrence Livermore Laboratory in Livermore, Calif., one of the sites affected by the intrusion. "The cost is going to be staggering."

Clifford Stoll, a computer security expert at Harvard University, added: "There is not one system manager who is not tearing his hair out. It's causing enormous headaches."

The affected computers carry a tremendous variety of business and research information among military officials, researchers and corporations.

While some sensitive military data are involved, the computers handling the nation's most sensitive secret information, like that on the control of nuclear weapons, are thought not to have been touched by the virus.

Parallel to Biological Virus

Computer viruses are so named because they parallel in the computer world the behavior of biological viruses. A virus is a program, or a set of instructions to a computer, that is either planted on a floppy disk meant to be used with the computer or introduced when the computer is communicating over telephone lines or data networks with other computers.

The programs can copy themselves into the computer's master software, or operating system, usually without calling any attention to themselves. From there, the program can be passed to additional computers.

Depending upon the intent of the software's creator, the program might cause a provocative but otherwise harmless message to appear on the computer's screen. Or it could systematically destroy data in the computer's memory. In this case, the virus program did nothing more than reproduce itself rapidly.

The program was apparently a result of an experiment, which

Continued on Page A21, Column 2

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Inquiry Shows Routine Billing of Government by Industry on Fees, Some Dubious

By JOHN H. CUSHMAN Jr.
Special to The New York Times

WASHINGTON, Nov. 3 — A Pentagon investigation has found that the nation's largest military contractors routinely charge the Defense Department for hundreds of millions of dollars paid to consultants, often without justification.

The report of the investigation said that neither the military's current rules nor the contractors' own policies are adequate to assure that the Government does not improperly pay for privately arranged consulting work. Senior Defense Department officials said the Pentagon was proposing changes to correct the flaws.

While it is not improper for military contractors to use consultants in performing work for the Pentagon, the work must directly benefit the military if it is to be paid for by the Defense Department. Often, Pentagon investigators discovered, this test is not met.

Broader Look at Consultants

The Justice Department's continuing criminal investigation has focused attention on consultants and their role in the designing and selling of weapons, and the Defense Department has been criticized for using consultants too freely. Now the Pentagon's own inves-

Registration Off Since 1984 Vote

There has been a pronounced decline in the percentage of eligible Americans who are registered to vote, a research group reports.

Nationally, the percentage of eligible Americans who are registered is estimated to be 78.9 percent, down 2.2 points from the 1984 level.

The group's study concluded that in many of the 30 states where final figures are available the decline was among

"All the News That's Fit to Print"

VOL. CXXXVIII . . . No. 47,679

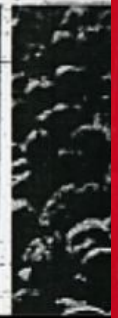
Gov. Michael S. Dukakis having his picture taken
a 10-year-old fan at a town meeting in Fairless Hi
Pa., during a tour of the Northeast in which he e
phasized the drug problem. Page A19. Vice Pr

Registration Off Since 1984 Vote

There has been a pronounced decline in the percentage of eligible Americans who are registered to vote, a research group reports.

Nationally, the percentage of eligible Americans who are registered is estimated to be 78.9 percent, down 2.2 points from the 1984 level.

The group's study concluded that in many of the 30 states where final figures are available the decline was among



Late Edition

New York: Today, partly sunny, milder. High 59-64. Tonight, mostly cloudy. Low 48-54. Tomorrow, cloudy, windy, rain developing. High 57-62. Yesterday: High 58, low 41. Details, page D16.

35 CENTS

PENTAGON REPORTS IMPROPER CHARGES FOR CONSULTANTS

CONTRACTORS CRITICIZED

Inquiry Shows Routine Billing of Government by Industry on Foes, Some Dubious

By JOHN H. CUSHMAN Jr.
Special to The New York Times

WASHINGTON, Nov. 3 — A Pentagon investigation has found that the nation's largest military contractors routinely charge the Defense Department for hundreds of millions of dollars paid to consultants, often without justification.

The report of the investigation said that neither the military's current rules nor the contractors' own policies are adequate to assure that the Government does not improperly pay for privately arranged consulting work. Senior Defense Department officials said the Pentagon was proposing changes to correct the flaws.

While it is not improper for military contractors to use consultants in performing work for the Pentagon, the work must directly benefit the military if it is to be paid for by the Defense Department. Often, Pentagon investigators discovered, this test is not met.

Broader Look at Consultants

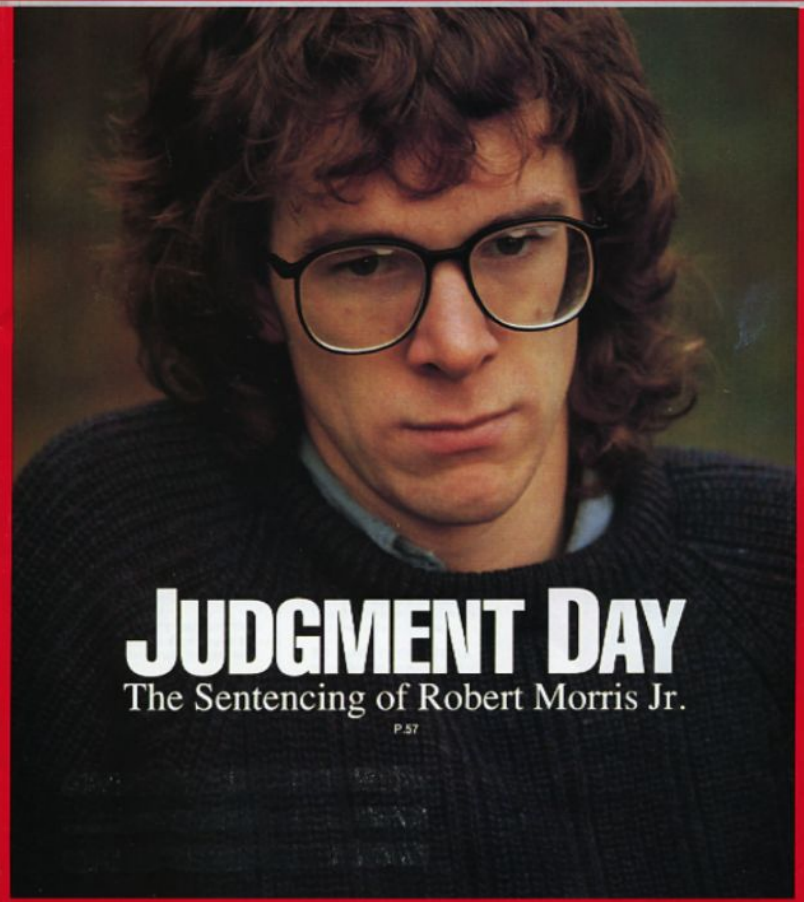
The Justice Department's continuing criminal investigation has focused attention on consultants and their role in the designing and selling of weapons, and the Defense Department has been criticized for using consultants too freely. Now the Pentagon's own inves-

MAY 7, 1990

INFORMATIONWEEK

THE NEWSMAGAZINE FOR INFORMATION MANAGEMENT

A CMP PUBLICATION $3.00



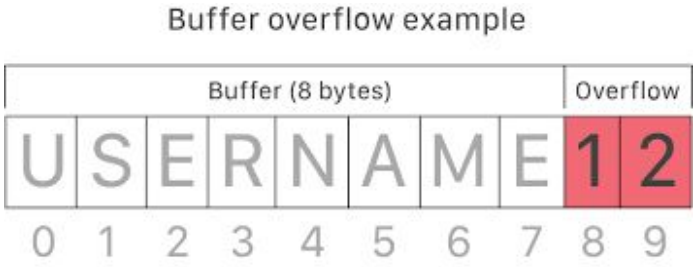
JUDGMENT DAY

The Sentencing of Robert Morris Jr.

P.57

How to take down the internet (in 1988)

1. Many programs were not “memory-safe” back then.
   a. Programs would let you access memory on the computer that you shouldn’t have access to
2. Find an array/buffer that lets you access memory you shouldn’t have access to.



3. Inject some malicious code right after that array.
   a. The computer will get tricked into running the code.
4. Accidentally add a bug that eats up all of the memory on each host computer.
5. Crash the entire internet.


The Morris Internet Worm source code
This disk contains the complete source code of the Morris Internet worm program. This tiny, 99-line program brought large pieces of the Internet to a standstill on November 2nd, 1988.
The worm was the first of many intrusive programs that use the Internet to spread.
Computer History Museum
Internet Worm - Source code
X1294.96A-D

"Responsible" Hacking

- The story of Robert Morris and his Internet Worm illustrates the core dilemma at the heart of security research
- Identifying and exposing security vulnerabilities is very important!
- Exposing security vulnerabilities in an irresponsible manner can result in devastating damages (monetary, physical, etc.)
- Responsible Disclosure: a vulnerability disclosure model in which a vulnerability or an issue is **disclosed only after a period of time that allows for the vulnerability or issue to be patched or mended**.

Back to our regularly scheduled programming...

Memory from the Stack vs. Heap

Memory from the Stack vs. Heap

```
Vector<string> varOnStack;
```

- Until today, all variables we've created get defined on the **stack**
- This is called static memory allocation
- Variables on the stack are stored directly to the memory and access to this memory is very fast
- We don't have to worry about memory management

Memory from the Stack vs. Heap

```
Vector<string> varOnStack;
```

- Until today, all variables we've created get defined on the **stack**
- This is **static** memory allocation
- Variables on the stack are stored directly to the memory and access to this memory is very fast
- We don't have to worry about memory management

```
string* arr = new string[numValues];
```

- We can now request memory from the **heap**
- This is **dynamic** memory allocation
- We have more control over variables on the heap
- But this means that we also have to handle the memory we're using carefully and properly clean it up when done

Cleaning Up

Cleaning Up

- When declaring local variables or parameters, C++ will automatically handle memory allocation and deallocation for you.

Cleaning Up

- When declaring local variables or parameters, C++ will automatically handle memory allocation and deallocation for you.
  - Memory allocation is the process by which the computer hands you a piece of computer memory in which you can store data.

Cleaning Up

- When declaring local variables or parameters, C++ will automatically handle memory allocation and deallocation for you.
  - Memory allocation is the process by which the computer hands you a piece of computer memory in which you can store data.
  - Memory deallocation is the process by which control of this memory (data storage location) is relinquished back to the computer

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.

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.

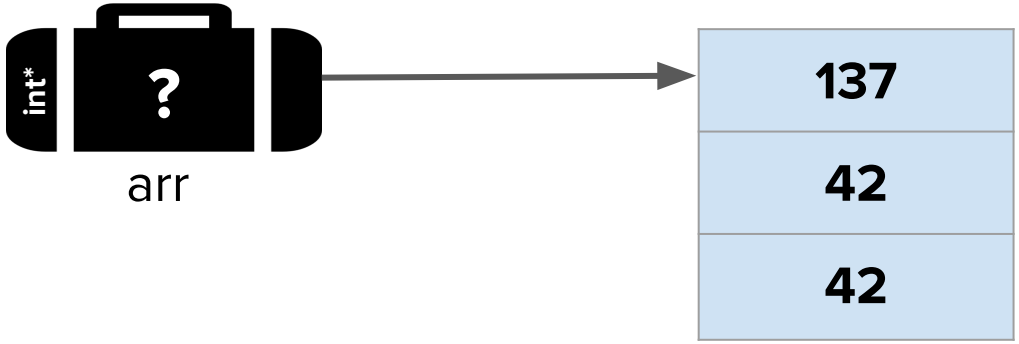
- If you don't, you get a **memory leak**. Your program will never be able to use that memory again.
  - Too many leaks can cause a program to crash – it's important to not leak memory!

Cleaning Up

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

```
delete[] arr;
```

- This destroys the array pointed to by the given pointer, not the pointer itself.
  - You can think of this operation as relinquishing control over the memory back to the computer.

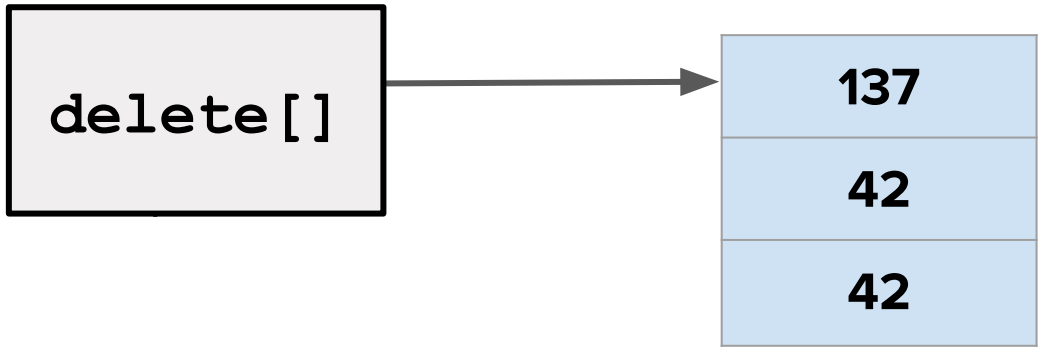


Cleaning Up

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

```
delete[] arr;
```

- This destroys the array pointed to by the given pointer, not the pointer itself.
  - You can think of this operation as relinquishing control over the memory back to the computer.

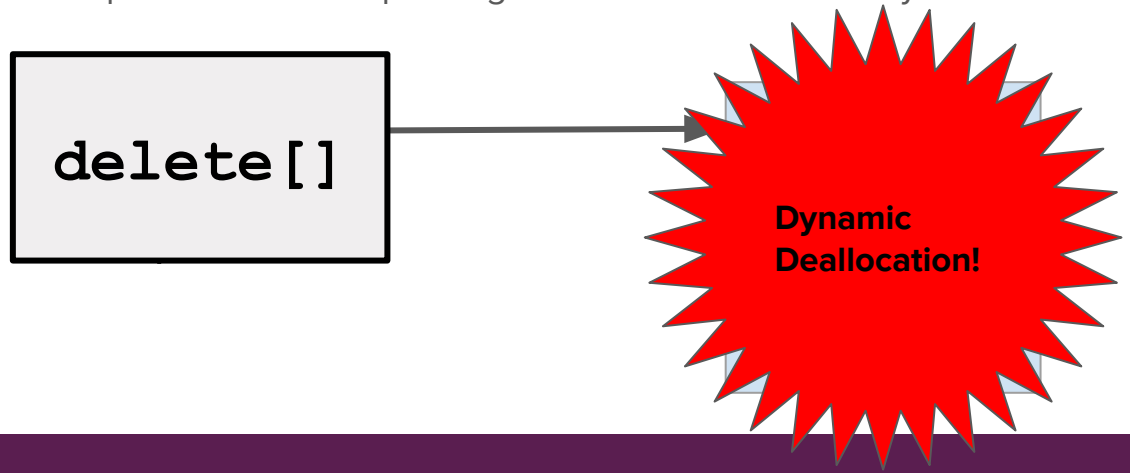


Cleaning Up

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

```
delete[] arr;
```

- This destroys the array pointed to by the given pointer, not the pointer itself.
  - You can think of this operation as relinquishing control over the memory back to the computer.

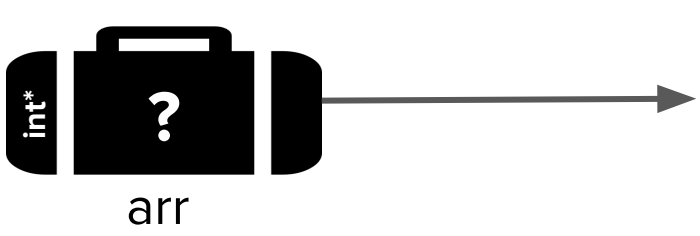


Cleaning Up

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

```
delete[] arr;
```

- This destroys the array pointed to by the given pointer, not the pointer itself.
  - You can think of this operation as relinquishing control over the memory back to the computer.

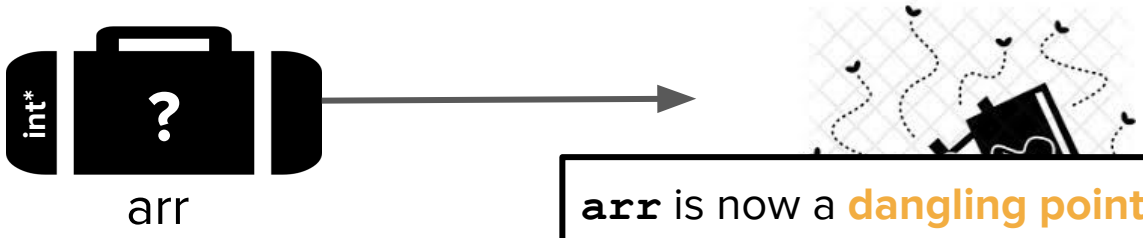


Cleaning Up

- You can deallocate (free) memory with the **`delete[]`** operator:

```
delete[] arr;
```

- This destroys the array pointed to by the given pointer, not the pointer itself.
  - You can think of this operation as relinquishing control over the memory back to the computer.



`arr` is now a **dangling pointer**. We can re-assign it to point somewhere else, but if we try to read from it or write to it, very bad, bad things will happen!

Takeaways

- You can create arrays of a fixed size at runtime by using **`new[]`**.
- C++ arrays don’t know their lengths and have no bounds-checking. With great power comes great responsibility.
- You are responsible for freeing any memory you explicitly allocate by calling **`delete[]`**.
- Once you’ve deleted the memory pointed at by a pointer, you have a dangling pointer and shouldn’t read or write from it.

Summary

Dynamic Memory and Arrays

- We’ve learned about **classes**, which have an **interface** and **implementation**.

Dynamic Memory and Arrays

- We've learned about **classes**, which have an **interface** and **implementation**.

- When implementing classes at the *lowest level of abstraction*, we need to use **dynamic memory** as a fundamental building block for specifying how much memory something needs.
  - We use the keyword **new** to allocate dynamic memory.
  - We keep track of that memory with a **pointer**. (more on pointers next week!)
  - We must clean up the memory when we're done with **delete**.

Dynamic Memory and Arrays

- We've learned about **classes**, which have an **interface** and **implementation**.

- When implementing classes at the *lowest level of abstraction*, we need to use **dynamic memory** as a fundamental building block for specifying how much memory something needs.
  - We use the keyword **new** to allocate dynamic memory.
  - We keep track of that memory with a **pointer**. (more on pointers next week!)
  - We must clean up the memory when we're done with **delete**.

- So far, we've learned how to allocate dynamic memory using **arrays**, which give us a contiguous block of memory that all stores one particular type (int, string, double, etc.).

What’s next?

Get ready to build a vector!

Arrays vs. Vectors

- Arrays are a very necessary tool to use if we want to actually store information in a structured way in a program.

- Vectors are a great abstraction, providing helpful methods and a clean interface that other programmers can use to solve interesting problems.

- **Idea:** Let's use a dynamically allocated array as the underlying method of data storage for a Vector class. Best of both worlds!

Implementing a Dynamic ADT