Dynamic Memory and Arrays

What are real-world examples of classes and abstractions?
Roadmap

Object-Oriented Programming

- arrays
- dynamic memory management
- linked data structures

Life after CS106B!

C++ basics

Core Tools

- testing
- diagnostic analysis

User/client

- vectors + grids
- stacks + queues
- sets + maps

Recursive problem-solving

Real-world algorithms

Implementation
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
**Definition**

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

How do we accomplish this in C++? With *classes*!
A class defines a new data type for our programs to use.
encapsulation
The process of grouping related information and relevant functions into one unit and defining where that information is accessible
What is a class?

- Examples of classes we’ve already seen: Vectors, Maps, Stacks, Queues

- 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).
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.
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.
Structs vs. classes (BankAccount)

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

class BankAccount {
public:
    BankAccount(string name, double amount);
    void deposit(double depositAmount);
    void withdraw(double withdrawalAmount);
    void transfer(double transferAmount, BankAccount& recipient);
    double getAmount() const;
    string getName() const;

private:
    string name;
    double amount;
};
```
Structs vs. classes (BankAccount)

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

```cpp
class BankAccount {
public:
    BankAccount(string name, double amount);
    void deposit(double depositAmount);
    void withdraw(double withdrawalAmount);
    void transfer(double transferAmount, BankAccount& recipient);
    double getAmount() const;
    string getName() const;
private:
    string name;
    double amount;
};
```

Better encapsulation! Error checking + limitations!

Controlled access!

No direct access to private data!
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.
Where are we now?
classes
object-oriented programming

abstract data structures (vectors, maps, etc.)

arrays
dynamic memory management
linked data structures

testing
algorithmic analysis
recursive problem-solving
Abstract data structures (vectors, maps, etc.)

Arrays
  - Dynamic memory management
  - Linked data structures
We've now crossed the abstraction boundary!

abstract data structures (vectors, maps, etc.)

dynamic memory management
linked data structures
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;
};
Turtles All the Way Down?
Turtles All the Way Down?

- Last time, we implemented a RandomBag on top of our library Vector type.
Turtles All the Way Down?

- Last time, we implemented a \texttt{RandomBag} on top of our library \texttt{Vector} type.

- But the \texttt{Vector} type is itself an abstraction (provided library) – what is it layered on top of?
Turtles All the Way Down?

- Last time, we implemented a **RandomBag** on top of our library **Vector** type.

- But the **Vector** type is itself an abstraction (provided 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?
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.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
</table>
Dynamically Allocating Arrays
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. \( int^* \), \( string^* \), \( Vector<\text{double}>^* \)
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.
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. \( \text{int}^* \), \( \text{string}^* \), \( \text{Vector<double>}^* \)
- Then, create a new array with the \textbf{new} keyword and assign the pointer to point to it.
- In two separate steps:

  ```cpp
  T* arr;
  arr = new T[size];
  ```
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:
  ```cpp
  T* arr;
  arr = new T[size];
  ```
- Or, in the same line:
  ```cpp
  T* arr = new T[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

● **Mid-quarter diagnostic**
  ○ 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 diagnostic 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 ½ the missed points.**
    - To participate: submit a **brief reflection (2-3 sentences is fine) on areas you want to focus on post-diagnostic** to the “Mid-Quarter Check-In” assignment on Paperless. Then use the IG Scheduling feature to sign up for time slot with your SL.

● **Assignment 4 is due Wednesday, July 28 at 11:59pm with a 24-hour grace period.** Assignment 5 will be released by end-of-day Wednesday.

● **Final project guidelines coming soon!**
Dynamic Allocation
Example
```cpp
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;
    }
}
```
```c++
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;
    }
}
```
```c++
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;
    }
}
```
```cpp
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 = 7
arr
```cpp
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;
    }
}
```
```c++
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;
    }
}
```
```cpp
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;
    }
}
```

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;
    }
}
```cpp
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;
    }
}
```
```cpp
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 = 7
arr = 0x8084ffff
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("Enter a string: ");
    }
    for (int i = 0; i < numValues; i++) {
        cout << i << " : " << arr[i] << endl;
    }
}
```c++
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;
    }
}
```
```cpp
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;
    }
}
```
```c++
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;
    }
}
```c++
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;
    }
}
```
```cpp
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;
    }
}
```

We Can

<table>
<thead>
<tr>
<th>Index:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>numValues</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arr</td>
<td>0x8084ffff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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;
    }
}

We Can Dance
Index: 0 1 2 3 4 5 6
```c++
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: 7
arr: 0x8084ffff
i: 2
```

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>We</td>
<td>Can</td>
<td>Dance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Index: 0 1 2 3 4 5 6
```cpp
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
```

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>We</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>0x8084ffff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```c++
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;
    }
}
```
```cpp
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;
    }
}
```
```cpp
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;
    }
}
```

We Can Dance If

<table>
<thead>
<tr>
<th>Index</th>
<th>We</th>
<th>Can</th>
<th>Dance</th>
<th>If</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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;
    }
}

We Can Dance If

Index: 0 1 2 3 4 5 6

numValues 7
arr 0x8084ffff
i 4
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 = 7
arr[0] = "We Can Dance"
arr[1] = "If"
arr[2] = "0x8084ffff"
arr[3] = "int 4"
index: 0 1 2 3 4 5 6
```cpp
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;
    }
}
```
```cpp
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
}
```

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>We</td>
<td>Can</td>
<td>Dance</td>
<td>If</td>
<td>We</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```cpp
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;
    }
}
```

```
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
```
```cpp
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;
    }
}
```
```c++
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;
    }
}
```
```c++
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;
    }
}
```

Index:

<table>
<thead>
<tr>
<th></th>
<th>We</th>
<th>Can</th>
<th>Dance</th>
<th>If</th>
<th>We</th>
<th>Want</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```cpp
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: 7
arr: 0x8084ffff
i: 6

<table>
<thead>
<tr>
<th>Index</th>
<th>We</th>
<th>Can</th>
<th>Dance</th>
<th>If</th>
<th>We</th>
<th>Want</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>We</td>
<td>Can</td>
<td>Dance</td>
<td>If</td>
<td>We</td>
<td>Want</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```c++
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;
    }
}
```

```
7
```

```
0x8084ffff
```

```
6
```

Index:
```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>We</td>
<td>Can</td>
<td>Dance</td>
<td>If</td>
<td>We</td>
<td>Want</td>
<td>To</td>
</tr>
</tbody>
</table>
```
```c++
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
arr
i

<table>
<thead>
<tr>
<th>We</th>
<th>Can</th>
<th>Dance</th>
<th>If</th>
<th>We</th>
<th>Want</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Index: 0 1 2 3 4 5 6 7
```cpp
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;
    }
}
```
```cpp
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;
    }
}
```

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>We</td>
<td>We</td>
<td>Can</td>
<td>Dance</td>
<td>If</td>
<td>We</td>
<td>Want</td>
<td>To</td>
</tr>
</tbody>
</table>

Example output:
```
0: We
1: Can
2: Dance
3: If
4: We
5: Want
6: To
```
```cpp
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;
    }
}
```
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++'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. Why is this 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.

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...
'Virus' in Military Computers Disrupts Systems Nationwide

By JOHN MARKOFF

In an innovation that raises questions about the vulnerability of the nation's computers, a Department of Defense network has been seized by a virus program apparently introduced by a computer scientist a week ago. The program reproduced itself through the computer networks, making hundreds of copies of itself in each machine it reached, effectively bringing systems linking thousands of military, corporate and university computers around the nation to a grinding halt. The virus is thought not to have destroyed any files.

By last Wednesday afternoon, computer experts were calling the virus the largest assault ever on the nation's computers.

The Big Issue

"The big issue is how a relatively benign software program can cause such damage to our computing community," said a computer security manager at Lawrence Livermore Laboratory.

"The cost is going up astronomically," said Clifford Smith, a computer scientist at Harvard University. "There is not one system manager who is not losing his hair yet. It's causing immense headaches.

"The affected computers carry a tremendous variety of business and research information among military officials, researchers and corporations. Some sensitive military data are involved, the computer scientists handling them are scrambling to identify secret vulnerabilities that could lead to the control of nuclear weapons, a thought that has been evoked by the virus.

Paradis in Biological Virus

Computer viruses are so named because they parallel in the computer world the behavior of biological organisms. A virus is a program, or a set of instructions to a computer, that is either placed 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 memory, to run or operate on its own, without calling on the attention of the user. From there, the programs can be passed on to other computers.

Depending on the state of the software's creator, the programs might cause a person to lose a program file, or otherwise harm the information stored on the computer. The virus could systematically destroy data in the computer's memory. In this case, the virus program did not produce any damage itself.
JUDGMENT DAY
The Sentencing of Robert Morris Jr.
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.
"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 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

string* arr = new string[numValues];

- We can now request memory from the heap
- This is called 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:
  
  ```cpp
  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?
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