Implementing Abstractions
Midterm

- CS106B Midterm Grading party tonight!
- Sorry for how poorly written the bundles question was =(  
  - I'll do my best to word things more clearly on the second midterm
From Last Time...
Where are We...

- Course Goal: Develop a strong understanding of basic data structures
- Class so far:
  - Week 1: Basic C++
  - Week 2: Data structures
  - Week 3: Recursion
  - Week 4: Algorithmic Analysis
We are *almost* ready to start implementing and analyzing data structures!

A couple C++ language features we need to cover.
Classes in C++

- Defining a class in C++ (typically) requires two steps:
  - Create a **header file** (typically suffixed with `.h`) describing the class's member functions and data members.
  - Create an **implementation file** (typically suffixed with `.cpp`) that contains the implementation of all the class's member functions.
- Clients of the class can then include the header file to use the class.
Getting Storage Space

• How do the Vector, Stack, Queue, etc. get space to store all the elements that they hold?
• C++ code can request extra storage space as the program is running.
• This is called dynamic memory allocation.
What is Memory?

• All variables and objects in C++ need somewhere to live inside the computer's memory.
  • This is RAM, by the way, not disk space.
• Whenever an object is created, space needs to be reserved for it.
Draw Memory
(Board)
New Stuff...
Memory Addresses

- Every object in C++ is physically located somewhere in memory.
- The location is called its address.
- Intuitively, think of the address as a link to the object, or a phone number for the object, or a name for the object.
- Given a variable, you can obtain its address by using the address-of operator (&):

```cpp
    cout << &myVariable << endl;
```
Pointers

• A **pointer** is a C++ variable that stores the address of an object.

• Given a pointer to an object, we can get back the original object.
  • Can then read the object's value.
  • Can then write the object's value.

• Think of a pointer as a URL for the object.
Choosing What to Point To

- Pointers store addresses, so if we want our pointer to point at an object, we can assign the pointer to the address of that object.
- For example:
  ```c
  int *myPtr = &myVariable;
  ```
- The object being pointed at is called the **pointee**.
Using a Pointer

• Once we have a pointer that points at some object, we can dereference the pointer to read and write that object.

• To dereference a pointer, prefix it with a *, as shown here:

```cpp
*ptr = 137;
cout << *ptr << endl;
```
Pointers, Visually

```c
int m = 137;
int n = 42;
```
int m = 137;
int n = 42;
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
*ptr1 = 2718;
*ptr2 = *ptr1;
ptr1 = ptr2;
*m = m / n;
```
int m = 137;
int n = 42;

int* ptr1 = &m;

*ptr1 = 2718;
*ptr2 = *ptr1;
ptr1 = ptr2;

m
137
ptr1

n
42
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;
```

```
137
  ↑
 ptr1
```

```
42
  ↑
 m
```

```
n
  ↑
```

```
ptr1
```

```
m / n = 3.2619
```
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;
```

![Diagram of pointers and variables]
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;

*ptr2 = *ptr1;
ptr1 = ptr2;
```

```
  m
  137
```
```
  n
  42
```
```
ptr1
```
```
ptr2
```
Pointers, Visually

\begin{verbatim}
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
\end{verbatim}
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```

```
+-----+      +-----+
| m   |      | n   |
+-----+      +-----+
    |      |      |
    v      v      v
+-----+      +-----+
| ptr1|      | ptr2|
+-----+      +-----+
```

```
2718

m

ptr1
```

```
42

n

ptr2
```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;

ptr1 = ptr2;

*ptr1 = m / n;
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;

ptr1 = ptr2;
*ptr1 = m / n;
```

![Diagram of pointers](image)
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;

ptr1 = ptr2;

*ptr1 = m / n;
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```

Diagram:
- `m` pointing to 2718
- `n` pointing to 42
- `ptr1` pointing to `m`
- `ptr2` pointing to `n`
Pointers, Visually

int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;

ptr1

ptr2
**Pointers, Visually**

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;

ptr1 = ptr2;

*m

n

2718

ptr1

ptr2
```
```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;

m = 160;
```

---

**Diagram:**
- **m** with value 2718
- **n** with value 2718
- **ptr1** pointing to **m**
- **ptr2** pointing to **n**
- **m** and **ptr1** are updated to 160 after assignment.
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;
```

![Diagram showing the relationship between pointers and their values]
**Pointers, Visually**

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
Pointers, Visually

```c
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
```

Assigning one pointer to another changes which object is being pointed at. It does not change the value of the pointee.
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
And finally, the reason we care about pointers...
Dynamic Memory Allocation
```c++
int main() {
  int numValues = getInteger("How many lines? ");

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

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

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

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

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

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

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

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

numValues = 7
int main() {
    int numValues = getInteger("How many lines? ");
    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getline();
    }
    for (int i = 0; i < numValues; i++) {
        cout << i << " : " << arr[i] << endl;
    }
}
```cpp
int main() {
    int numValues = getInteger("How many lines? ");
    string* arr = new string[numValues];
    for (int i = 0; i < numValues; i++) {
        arr[i] = getline();
    }
    for (int i = 0; i < numValues; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}
```
```cpp
int main() {
    int numValues = getInteger("How many lines? ");

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

numValues 7 arr  i 1
```cpp
int main() {
    int numValues = getInteger("How many lines? ");

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

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

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

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

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

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

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

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

We Can Dance
```cpp
int main() {
    int numValues = getInteger("How many lines? ");

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

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

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

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

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

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

numValues 7 arr 3
int main() {
    int numValues = getInteger("How many lines? ");

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

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

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

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

We Can Dance If
```c++
int main() {
    int numValues = getInteger("How many lines? ");

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

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

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

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

We Can Dance If We
int main() {
    int numValues = getInteger("How many lines? ");

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

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

We Can Dance If We
int main() {
    int numValues = getInteger("How many lines? ");

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

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

numValues 7  arr  i  5

We Can Dance If We
int main() {
    int numValues = getInteger("How many lines? ");

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

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

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

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

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

We Can Dance If We Want
int main() {
    int numValues = getInteger("How many lines? ");

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

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

We Can Dance If We Want
int main() {
    int numValues = getInteger("How many lines? ");

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

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

We Can Dance If We Want

numValues 7 arr 6 i

We Can Dance If We Want
int main() {
    int numValues = getInteger("How many lines? ");

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

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

    numValues 7  arr  6

    We Can Dance If We Want 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();
    }

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

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

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

We Can Dance If We Want To
int main() {
    int numValues = getInteger("How many lines? ");

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

    for (int i = 0; i < numValues; i++) {
        cout << i << " : 
    }
}

numValues 7  arr

We Can Dance If We Want 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();
    }

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

We Can Dance If We Want To
int main() {
    int numValues = getInteger("How many lines? ");

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

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

We Can Dance If We Want To
Dynamically Allocating Arrays

• First, declare a variable that will point at the newly-allocated array. If the array elements have type $T$, the pointer will have type $T^*$. 

• Then, create a new array with the \texttt{new} keyword and assign the pointer to point to it.

• In two separate steps:

  
  $T^* \ arr; \\
  arr = \texttt{new} \ T[size];$

• Or, in the same line:

  
  $T^* \ arr = \texttt{new} \ T[size];$
Cleaning Up

• When declaring global variables or local variables, 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* and will slowly exhaust all of memory.

• Eventually, the program will crash when you ask for more memory with `new`, because the program thinks all of memory is in use.
Memory.cpp
(Memory and Leaking Memory)
Cleaning Up

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

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

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

  ```c++
  delete[] ptr;
  ```

• This destroys the array pointed at by the given pointer, not the pointer itself.
Cleaning Up

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

  \[
  \text{delete[]} \text{ ptr} ;
  \]

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

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

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

- C++ has few of the safety features present in Java.
- All of the following result in *undefined behavior* in C++:
  - Reading or writing through a pointer that you haven't initialized.
  - Reading or writing through a pointer to an array that you have deallocated.
  - Reading or writing off the end of an array.
Implementing Stack
Implementing Stack

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

• A bounded stack.
• Allocate a fixed-size array for elements.
• Add elements to the array when they're pushed.
• Remove elements from the array when they're popped.
• Report an error if we exceed the size of the array.
An Initial Idea

- element array
- allocated length: 4
- logical length: 0
An Initial Idea

- **element array**
- **allocated length**: 4
- **logical length**: 1

The diagram shows a box labeled "137" connected to a box with three elements: "element array", "allocated length", and "logical length". The allocated length is 4, and the logical length is 1.
An Initial Idea

- Element array
- Allocated length: 4
- Logical length: 2

137 42
An Initial Idea

```
137 42 2718
```

- element array
- allocated length: 4
- logical length: 3
An Initial Idea

<table>
<thead>
<tr>
<th>element array</th>
<th>allocated length</th>
<th>logical length</th>
</tr>
</thead>
<tbody>
<tr>
<td>137 42 2718 512</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

137 42 2718 512
An Initial Idea

- element array
- allocated length: 4
- logical length: 3
- Array: 137, 42, 2718, 512
An Initial Idea

element array
allocated length
logical length

137 42 2718 512
An Initial Idea

- Element array
- Allocated length: 4
- Logical length: 3

137 42 161 512
An Initial Idea

137  42  161  314

- element array
- allocated length: 4
- logical length: 4
What methods does our Stack need?
What data does our Stack need?
Let's Code it Up!
Constructors

• A **constructor** is a special member function used to set up the class before it is used.

• The constructor is automatically called when the object is created.

• Syntax: The constructor for a class named **ClassName** has signature

  ```
  ClassName(args);
  ```
Destructors

- A destructor is a special member function responsible for cleaning up an object's memory.
- Automatically called when a local variable goes out of scope.
- Automatically called if you delete a pointer to an object.
- Syntax: The destructor for a class named `ClassName` has signature
  
  ```
  ~ClassName () ;
  ```
Next Time

- **Making Stack Grow!**
  - Different approaches to Stack growth.
  - Analysis of these approaches.