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

private:
    Vector<int> elems;
};
class RandomBag {
public:
    void add(int value);
    int removeRandom();
    int size() const;
    bool isEmpty() const;

private:
    Vector<int> elems;
};
Turtles All the Way Down?

• Last time, we implemented a `RandomBag` on top of our library `Vector` type.

• But the `Vector` type is itself a library – what is it layered on top of?

• **Question:** What are the fundamental building blocks provided by the language, and how do we use them to build our own custom classes?
## Getting Storage Space

- The Vector, Stack, Queue, etc. all need storage space to put the elements that they store.
- That storage space is allocated using *dynamic memory allocation*.
- Essentially:
  - You can, at runtime, ask for extra storage space, which C++ will give to you.
  - You can use that storage space however you’d like.
  - You have to explicitly tell the language when you’re done using the memory.
Dynamic Allocation Demo
```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;
    }
}
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
```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;
    }
    numValues = 7;  // Change the numValues to 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();
    }
    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;
    }
}
```

Because the variable `arr` points to the array, it is called a `pointer`. 
```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;
    }
}
```
```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;
    }
}
```
```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
int main() {
  int numValues = getInteger("How many lines? ");

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

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

numValues 7 arr i 2

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

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

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

We Can Dance

numValues 7 arr i 2
```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;
    }
}
```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
```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;
    }
}
```
```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
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 4
```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
7 numValues 7 arr 5 i
```
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
```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 <<": "; cout << 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;
    }
}
```

```
We Can Dance If We Want To 7 numValues 7 arr i
```
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 $T^*$.  
  - e.g. `int*`, `string*`, `Vector<double>*`
- Then, create a new array with the `new` keyword and assign the pointer to point to it.
- In two separate steps:
  ```
  T* arr;
  arr = new T[size];
  ```
- Or, in the same line:
  ```
  T* arr = new T[size];
  ```
Dynamically Allocating Arrays

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

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

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

  ```cpp
  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:

```cpp
delete[] ptr;
```

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

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

```
delete[] ptr;
```

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

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

- 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.
Time-Out for Announcements!
TREEHACKS 2019
FEBRUARY 15 - 17

HACKING YOU CAN'T MISS.

Join us for 36 straight hours of hacking with hundreds of others from across the nation. We'll provide the space, mentors, workshops, and tech that you need to build anything your heart desires, in addition to great food, performances, and people.

THREE NEW VERTICALS:
- HEALTH
- AWARENESS
- SECURITY

SIGN UP BY FEBRUARY 13:
APPLY.TREEHACKS.COM
Midterm Exam

• The midterm exam is next **Tuesday, February 19** from **7:00PM - 10:00PM**.
  • Location TBA.
• It covers topics from Lectures 01 – 12 (up through and including big-O notation) and Assignments 0 – 4.
• The exam is closed-book and limited-note. You may bring one double-sided sheet of 8.5” × 11” of notes to the exam with you.
• If you believe you will be taking the exam at an alternate time and have not yet heard from Kate, you are not actually taking the exam at an alternate time. Ping us ASAP. 😊
Midterm Exam

• We will be administering the exam using a software tool called BlueBook.

• Visit the CS106B website, click the “BlueBook” link under the “Resources” tab, then download the BlueBook software.

• This week’s section handout will be done through BlueBook so that you get a chance to test it out.

• Need a laptop for the exam? Feel free to contact us. We can help out with that.
Midterm Exam

- **We want you to do well on this exam.** The purpose of this exam is for you to show us what you’ve learned, not to separate the elect from the damned.

- We will be holding a practice midterm exam
  
  *Tomorrow,*
  
  **7PM - 10PM,** in
  
  **Room 320-105.**

- The practice exam will be administered via BlueBook. The practice exam files will be posted on the course website soon. We’ll release the password tomorrow at the practice exam.
continue;
Implementing Stack
Implementing Stack

• Last time, we saw how to implement RandomBag in terms of Vector.

• We could also implement Stack in terms of Vector.

• What if we wanted to implement the Stack without relying on any other collections?

• Let's build the stack directly!
You Gotta Start Somewhere

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

- Element array
- Allocated size: 4
- Logical size: 0
An Initial Idea

The stack’s allocated size is the number of slots in the array. Remember – arrays in C++ cannot grow or shrink.
An Initial Idea

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

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

- Element array
- Allocated size: 4
- Logical size: 1

137
An Initial Idea

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

137  42
An Initial Idea

- Element array: 137, 42, 2718
- Allocated size: 4
- Logical size: 3
An Initial Idea

- Element array
- Allocated size: 4
- Logical size: 4

Numbers: 137, 42, 2718, 512
An Initial Idea

137 42 2718 512

- element array
- allocated size
- logical size

4 3
Arrays cannot grow or shrink, so this older value is still technically there in the array. We’re just going to pretend it isn’t.
An Initial Idea

```
137  42  2718  512
```

- **element array**
- **allocated size**: 4
- **logical size**: 2
An Initial Idea

137  42  161  512

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

allocated size

logical size

element array

allocated size

logical size
What We Have
Before We Start: A Problem
Cradle to Grave

```c
int main() {
    OurStack stack;

    /* The stack lives a rich, happy, 
    * fulfilling life, the kind we 
    * all aspire to. 
    */

    return 0;
}
```
Cradle to Grave

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

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

Undefined behavior!

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

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

\[ \text{ClassName}(\text{args}) ; \]
Implementing our Operations
class OurStack {
    public:
        OurStack();
        int peek() const;
        void push(int value);
        int pop();
        int size() const;
        bool isEmpty() const;
    private:
        int* elems;
        int allocatedSize;
        int logicalSize;
};
So... we’re done?
Cradle to Grave, Take II

```c
int main() {
    OurStack stack;

    /* The stack lives a rich, happy, 
       * fulfilling life, the kind we 
       * all aspire to. 
       */

    return 0;
}
```
Cradle to Grave, Take II

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

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

allocated size

logical size
Cradle to Grave, Take II

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

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

```
```
Cradle to Grave, Take II

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

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

- element array
- allocated size: 4
- logical size: 0
int main() {
    OurStack stack;
    /* The stack lives a rich, happy, fulfilling life, the kind we all aspire to. */
    return 0;
}

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

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

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

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

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

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

• A destructor is a special member function responsible for cleaning up an object's memory.

• It’s automatically called whenever an object’s lifetime ends (for example, if it’s a local variable that goes out of scope.)

• The destructor for a class named ClassName has signature

   ~ClassName();
Cradle to Grave, Take III

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

    /* The stack lives a rich, happy, fulfilling life, the kind we all aspire to. */

    return 0;
}
Cradle to Grave, Take III

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

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

element array
allocated size
logical size

???
???
Cradle to Grave, Take III

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

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

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

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

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

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

OurStack::~OurStack() {
    delete[] elems;
}
```
Cradle to Grave, Take III

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

```c
int main() {
    OurStack stack;

    /* The stack lives a rich, happy, 
       * fulfilling life, the kind we 
       * all aspire to. 
       */
    return 0;
}
```
To Summarize

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

- **Download BlueBook and Section Handout 5.**
  - It’s a good idea to familiarize yourself with these tools before the actual exam.
  - Need a laptop? Ping us! It’s not a problem.
- **Keep working on Assignment 4.**
  - If you’re following our timetable, you should be mostly done with Disaster Preparations at this point and should start working on Winning the Presidency.
Next Time

• **Making Stack Grow!**
  • Different approaches to Stack growth.
  • Analysis of these approaches.
  • The reality: *everything is a tradeoff!*

• **Implementing the Queue**
  • ... is not too hard when you have a stack!