Welcome to C++!

Trip Master
Zheng Lian

-Professor Oak, CS106B alum and C++ pro
Welcome to YEAH

- Your Early Assignment Help was conceived many moons ago to help students start early on assignments.
- We’ll go over each part of the assignment, and I’ll give helpful tips / hints, as well as conceptual overviews for trickier topics.
  - Zheng and I will do our best to answer your questions :)
- These slides are **NOT** a supplement to reading the assignment handouts!
  - Keith’s explanations are far more comprehensive and insightful!
- If possible, please come to the live YEAH sessions. Zheng and I get lonely :(
About us

Trip Master

- Senior studying CS Systems (hoping to study Systems as a coterm soon)
- A Cappella nerd and teaching fanatic

Zheng Lian (right)

- Coterm studying AI (studied CS Theory as an undergrad)
- Loves to sing and photograph with farm animals
Assignment 1 Logistics

- The assignment is due on **Friday 1/22 at the start of class**
- This assignment must be completed **individually** (but hopefully you know that!)
Assignment Overview

- Stack Overflow
  - When recursion can go terribly wrong!
Assignment Overview

- Stack Overflow
  - When recursion can go terribly wrong!
- Only Connect
  - Recursive text modification!
Assignment Overview

- Stack Overflow
  - When recursion can go terribly wrong!
- Only Connect
  - Recursive text modification!
- Playing Fair
  - Fairness to a higher order
Assignment Overview

- Stack Overflow
  - When recursion can go terribly wrong!
- Only Connect
  - Recursive text modification!
- Playing Fair
  - Fairness to a higher order
- Sandpiles
  - Recursion is beautiful!
Assignment Overview

- Stack Overflow
  - When recursion can go terribly wrong!
- Only Connect
  - Recursive text modification!
- Playing Fair
  - Fairness to a higher order
- Sandpiles
  - Recursion is beautiful!
- Plotter
  - Iteration can be beautiful, too :)

![Image of Buzz Lightyear and Woody with text: RECURSION RECURSION EVERYWHERE]
Before We Begin

- Please let us know if there is any technical difficulty
- And please ask clarifying questions! They are extremely helpful to both you and your peers
Part 1: stack overflow

- Every time you call a function, a portion of your computer’s memory (DRAM) is allocated called a stack frame.
Part 1: stack overflow

- Every time you call a function, a portion of your computer’s memory (DRAM) is allocated called a stack frame.
- In a recursive function, every time we recurse, the computer needs to generate a new stack frame!

As you can see, each time `digitalRootOf()` calls itself, a new frame is created!
What happens if we don’t include a proper terminating (base) case in our code?

Is there a limit to how many stack frames we can create?
A program that has infinite recursion will run into something called **Stack Overflow**

- **Stack Overflow** happens when your computer runs out of RAM to allocate to your programs!
- Attempting to create infinite stack frames will do this!
Part 1: Stack Overflow

- A program that has infinite recursion will run into something called Stack Overflow
  - Stack Overflow happens when your computer runs out of memory (RAM) to allocate to your programs!
  - Attempting to create infinite stack frames will do this!

Stack Overflow problems happen all the time in the real world!
Part 1: Stack Overflow

- In this first part, your task is to examine the file `Stackoverflow.cpp` and step through it in the debugger.
- All you have to do is run the Stack Overflow program in the debugger.
  - Question for the reader: what's the difference between running this program in normal mode vs debug mode? (Try it!)
Once your program has crashed due to Stack Overflow, you’ll need to examine the call stack, the sequence of function calls that the program executed.
Part 1: stack overflow

- Once your program has crashed due to Stack Overflow, you’ll need to examine the call stack, the sequence of function calls that the program executed.
- You should notice a variable `index` appear in your debug window on the right side of your screen. (I’ve hidden its value)
Part 1: Stack Overflow

- Once your program has crashed due to Stack Overflow, you'll need to examine the call stack, the sequence of function calls that the program executed.
- You should notice a variable `index` appear in your debug window on the right side of your screen. (I've hidden its value)
- As you click on other instances in the call stack, you'll notice that the value of `index` will change!
Part 1: stack overflow

- Once your program has crashed due to Stack Overflow, you’ll need to examine the call stack, the sequence of function calls that the program executed.
- You should notice a variable `index` appear in your debug window on the right side of your screen. (I’ve hidden its value)
- As you click on other instances in the call stack, you’ll notice that the value of `index` will change!
- Your task is to identify the cycle in the values of `index`, which is causing the infinite recursion!
  - Report these numbers in the comments of the `Stackoverflow.cpp` file
Questions about stack overflow?

Never have I felt so close to another soul and yet so helplessly alone as when I Google an error and there's one result—a thread by someone with the same problem and no answer last posted to in 2003.

Who were you, DenveCoder9? What did you see?!
Before we continue, let's talk about...
Kanye StudentTest's Testing Overview!

That that don't can only make me stronger.

― Kanye StudentTest, CS106B alum and unit-testing pro
Running Tests in CS106B

- An important part of CS106B is testing, the ability to write small pieces of functionality that you can test.
- There are 4 functions you’ll be frequently using this quarter, TIME_OPERATION, EXPECT, EXPECT_EQUAL, and EXPECT_ERROR.
  - You will create STUDENT_TESTs and use TIME_OPERATION, EXPECT, EXPECT_EQUAL, and EXPECT_ERROR to verify the correctness of your functions!
- TIME_OPERATION (inputsize, operation) function call times how long it takes to perform function OPERATION on INPUTSIZE elements, and reports these numbers to the console.
- Check out the use of the EXPECT functions use on the next slide!
Running Tests in CS106B

```c
int returnFive (int num) {
    if (num == -1) error ("Error, cannot process -1!");
    return 5;
}

// EXPECT tests the provided predicate.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT") {
    EXPECT (returnFive (0) == 5);
}

// EXPECT_EQUAL compares two values.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT_EQUAL") {
    EXPECT_EQUAL (returnFive (0), 5);
}

// EXPECT_ERROR passes if and only if the code it runs throws an error.
STUDENT_TEST ("Verifies that returnFive throws an error on bad input with EXPECT_ERROR") {
    EXPECT_ERROR (returnFive (-1));
}
```
Questions about testing?

```c
int returnFive (int num) {
    if (num == -1) error ("Error, cannot process -1!");
    return 5;
}

// EXPECT tests the provided predicate.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT") {
    EXPECT (returnFive (0) == 5);
}

// EXPECT_EQUAL compares two values.
STUDENT_TEST ("Verifies that returnFive returns five with EXPECT_EQUAL") {
    EXPECT_EQUAL (returnFive (0), 5);
}

// EXPECT_ERROR passes if and only if the code it runs throws an error.
STUDENT_TEST ("Verifies that returnFive throws an error on bad input with EXPECT_ERROR") {
    EXPECT_ERROR (returnFive (-1));
}
```
Let's start coding!
Part 2: Only Connect

In this part of the assignment, you'll be asked to implement the following recursive function:

```cpp
string onlyConnectize(string phrase);
```

which, given a string of text, removes all characters except the consonants, and converts the remainder to uppercase.
Part 2: Only Connect

- In this part of the assignment, you'll be asked to implement the following recursive function:

  ```
  string onlyConnectize(string phrase);
  ```

  which, given a string of text, removes all characters except the consonants, and converts the remainder to uppercase.

  ```
  onlyConnectize("Elena Kagan") returns "LNKGN",
  onlyConnectize("Antonin Scalia") returns "NTNNSCL",
  onlyConnectize("EE 364A") returns "",
  onlyConnectize("For sale: baby shoes, never worn.") returns "FRSLBBSNSNVRWRN",
  onlyConnectize("Thank you, next (next)") returns "THNKNXTNXT", and
  onlyConnectize("Annie Mae, My Sea Anemone Enemy!") returns "NNMMSNMNNNM".
  ```
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- A few hints about this problem:
  - Notice that the return type of this function is `string`. What might this tell you about the base case for this function?
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- A few hints about this problem:
  - Notice that the return type of this function is `string`. What might this tell you about the base case for this function?
  - Think about this problem as going character by character.
Part 2: Only Connect

```cpp
template<typename T>
T onlyConnectize(T phrase);
```

- A few hints about this problem:
  - Notice that the return type of this function is `string`. What might this tell you about the base case for this function?
  - Think about this problem as going character by character.

What should you return if the first character is a vowel / non letter? *Do we even care about remembering this character?*
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- A few hints about this problem:
  - Notice that the return type of this function is `string`. What might this tell you about the base case for this function?
  - Think about this problem as going character by character.
  - There is no built-in `isvowel` function in the cpp libraries -- you’ll have to write your own.
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- A few hints about this problem:
  - Notice that the return type of this function is `string`. What might this tell you about the base case for this function?
  - Think about this problem as going `character by character`.
  - There is **no** built in `isvowel` function in the cpp libraries -- you’ll have to write your own.
  - Take a look at the Stanford library “strlib.h” for some ~helpful~ string functions.
    - From personal experience, the people who do the best in this class make use of all that the Stanford libraries have to offer!
Part 2: Only Connect

string onlyConnectize(string phrase);

● A few hints about this problem:
  ○ Notice that the return type of this function is string. What might this tell you about the base case for this function?
  ○ Think about this problem as going character by character.
  ○ There is no built in isvowel function in the cpp libraries -- you’ll have to write your own.
  ○ Take a look at the Stanford library “strlib.h” for some ~helpful~ string functions.
    ■ From personal experience, the people who do the best in this class make use of all that the Stanford libraries have to offer!
  ○ Here are a few helpful char functions to help you get started!

int isalpha ( int c );

int toupper ( int c );

Returns a nonzero value (equivalent to true) if the provided character is in the alphabet

Returns the uppercase version of the provided character (or the same character if invalid)
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- A few hints about this problem:
  - Notice that the return type of this function is `string`. What might this tell you about the base case for this function?
  - Think about this problem as going character by character.
  - There is no built-in `isvowel` function in the C++ libraries—you'll have to write your own.
  - Take a look at the Stanford library “strlib.h” for some ~helpful~ string functions.
    - From personal experience, the people who do the best in this class make use of all that the Stanford libraries have to offer!
  - Here are a few helpful char functions to help you get started!

But wait... why do these functions take in `ints` and not `chars`?

```cpp
int isalpha ( int c );
int toupper ( int c );
```

- Returns a nonzero value (equivalent to true) if the provided character is in the alphabet
- Returns the uppercase version of the provided character (or the same character if invalid)
It's time for...
Charole Baskin’s brief foray into char representation via the ASCII set!

-Charole Baskin, 106B alum and mariticide suspect
How do we represent characters?

- Let’s face it, there are a lot of unique chars out there. When you couple that with the existence of fonts, you get a data representation nightmare – how do you represent chars?
How do we represent characters?

- Let’s face it, there are a lot of unique chars out there. When you couple that with the existence of fonts, you get a data representation nightmare – how do you represent chars?

- The computing world decided to get together to create a standard number representation for popular chars (128 of them!). Each char would correspond to an integer in a table called the ASCII set.
How do we represent characters?

- Let’s face it, there are a lot of unique chars out there. When you couple that with the existence of fonts, you get a data representation nightmare – how do you represent chars?
- The computing world decided to get together to create a standard number representation for popular chars (128 of them!). Each char would correspond to an integer in a table called the ASCII set.
- For example, ‘A’ -> 65, and ‘a’ -> 97.
What does this code print?

```cpp
string s = "apple";
cout << toupper(s[0]) << endl;
```
How do we represent characters?

- You need to be careful that you’re not working directly with integers when you work with characters!
  - If a function returns an int, be sure you’re storing the data as a character so that it can be read properly!

```cpp
string s = "apple";
char firstLetter = toupper(s[0]);
cout << firstLetter << endl;
```
Back to the program...
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- Some final thoughts:
  - This function must be implemented recursively. No loops please!
Part 2: Only Connect

```
string onlyConnectize(string phrase);
```

- Some final thoughts:
  - This function must be implemented **recursively**. No loops please!
  - Feel free to add helper functions like `isVowel`! We love decomposition :)

Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- Some final thoughts:
  - This function must be implemented recursively. No loops please!
  - Feel free to add helper functions like `isVowel`! We love decomposition :)
  - Remember that we're treating ‘y’ like a vowel :)
Part 2: Only Connect

```cpp
string onlyConnectize(string phrase);
```

- Some final thoughts:
  - This function must be implemented **recursively**. No loops please!
  - Feel free to add helper functions like `isVowel`! We love decomposition :)
  - Remember that we're treating 'y' like a vowel :)
  - Be sure to add robust tests to your program to verify its correctness on tricky cases!
Questions about Only Connect?

string onlyConnectize(string phrase);

Confusion is nothing to be ashamed of -- just ask The Boulder!
Part 3: Playing Fair

- Next up, you’ll be writing two recursive functions that, given some integer $n$, produce either an A sequence or a B sequence of order $n$.

```c
string aSequenceOfOrder(int n);
string bSequenceOfOrder(int n);
```
Part 3: Playing Fair

- Next up, you’ll be writing two recursive functions that, given some integer \( n \), produce either an A sequence or a B sequence of order \( n \).
- Here’s what these sequences look like in various orders!

<table>
<thead>
<tr>
<th>Order 0</th>
<th>Order 1</th>
<th>Order 2</th>
<th>Order 3</th>
<th>Order 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-sequence:</td>
<td>A</td>
<td>AB</td>
<td>ABBA</td>
<td>ABBABAAB</td>
</tr>
<tr>
<td>B-sequence:</td>
<td>B</td>
<td>BA</td>
<td>BAAB</td>
<td>BAABABBA</td>
</tr>
</tbody>
</table>
Part 3: Playing Fair

- Next up, you’ll be writing two recursive functions that, given some integer \( n \), produce either an A sequence or a B sequence of order \( n \).
- Here’s what these sequences look like in various orders!

<table>
<thead>
<tr>
<th>Order 0</th>
<th>Order 1</th>
<th>Order 2</th>
<th>Order 3</th>
<th>Order 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-sequence:</td>
<td>A</td>
<td>AB</td>
<td>ABBA</td>
<td>ABBABAAB</td>
</tr>
<tr>
<td>B-sequence:</td>
<td>B</td>
<td>BA</td>
<td>BAAB</td>
<td>BAABABBA</td>
</tr>
</tbody>
</table>

```cpp
cout << aSequenceOfOrder(0) << endl;    // "A"
cout << bSequenceOfOrder(3) << endl;    // "BAABABBA"
```
Part 3: Playing Fair

- To get insight into solving this problem, it will help to think about building a solution from the ground up.

<table>
<thead>
<tr>
<th>Order 0</th>
<th>Order 1</th>
<th>Order 2</th>
<th>Order 3</th>
<th>Order 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-sequence:</td>
<td>A</td>
<td>AB</td>
<td>ABBA</td>
<td>ABBABAAB</td>
</tr>
<tr>
<td>B-sequence:</td>
<td>B</td>
<td>BA</td>
<td>BAAB</td>
<td>BAABABBA</td>
</tr>
</tbody>
</table>
To get insight into solving this problem, it will help to think about building a solution from the ground up.

- For example, an order-1 A sequence is simply an order-0 A sequence concatenated with an order-0 B sequence!

<table>
<thead>
<tr>
<th>Order 0</th>
<th>Order 1</th>
<th>Order 2</th>
<th>Order 3</th>
<th>Order 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-sequence:</td>
<td>A</td>
<td>AB</td>
<td>ABBA</td>
<td>ABBABAAB</td>
</tr>
<tr>
<td>B-sequence:</td>
<td>B</td>
<td>BA</td>
<td>BAAB</td>
<td>BAABABBA</td>
</tr>
</tbody>
</table>

- Does this pattern continue for an order-n sequence?
Part 3: Playing Fair

- It totally does!

This order-4 A sequence consists of an order-3 A sequence and an order-3 B Sequence!
Part 3: Playing Fair

- It totally does!

This order-3 B sequence consists of an order-2 B sequence and an order-2 A Sequence!
Part 3: Playing Fair

- Some more notes about this problem:
  - Beware of large inputs to this function -- as you can see, the length of a sequence is $2^n$ of the order, so summon large sequences at your own risk.
Part 3: Playing Fair

- Some more notes about this problem:
  - Beware of large inputs to this function -- as you can see, the length of a sequence is $2^n$ of the order, so summon large sequences at your own risk.
  - This function must be implemented recursively. None of that old-school loopy business.
    - Keep in mind that you are allowed to call the BSequence() function from the ASequence() function!
Part 3: Playing Fair

- Some more notes about this problem:
  - Beware of large inputs to this function -- as you can see, the length of a sequence is $2^n$ of the order, so summon large sequences at your own risk.
  - This function must be implemented recursively. None of that old-school loopy business.
    - Keep in mind that you are allowed to call the `BSequence()` function from the `ASequence()` function!
  - Recall that this function takes an integer input $n$. If the user inputs a negative number, you should raise an error via the following syntax!

```javascript
error("a string containing your error message");
```
Questions about Playing Fair?

```java
int getRandonNumber()
{
    return 4; // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Part 4: Sandpiles

- In this penultimate part, you’ll be writing code to simulate a sandbox like so:
Part 4: Sandpiles

- In this penultimate part, you’ll be writing code to simulate a sandbox like so:
- Each tile represents an entry in a `Grid<int>` called `world`. 
Part 4: Sandpiles

- In this penultimate part, you’ll be writing code to simulate a sandbox like so:
- Each tile represents an entry in a `Grid<int>` called `world`.
- More specifically, you’ll need to simulate the dropping of a grain of sand into this world via the recursive function:

  ```cpp
  void dropSandOn(Grid<int>& world, int row, int col);
  ```

  which attempts to drop a grain of sand into `world` at location `{row, col}`
Part 4: Sandpiles

Here’s how dropping sand works:

The sandbox will start looking empty!
Part 4: Sandpiles

Here’s how dropping sand works:

dropSandOn(world, 2, 2);

The sandbox will start looking empty!
Part 4: Sandpiles

Here's how dropping sand works:

```
dropSandOn(world, 2, 2);
```

Now we have a single element at location {2, 2}
Part 4: Sandpiles

Here’s how dropping sand works:

dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
Part 4: Sandpiles

Here’s how dropping sand works:

dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);

The number increases for every grain added!
Part 4: Sandpiles

Here’s how dropping sand works:

dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);

The number increases for every grain added!
Part 4: Sandpiles

Here’s how dropping sand works:

```java
dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
```

The number increases for every grain added!
Part 4: Sandpiles

Here’s how dropping sand works:

dropSandOn(world, 2,2);
dropSandOn(world, 2,2);
dropSandOn(world, 2,2);
dropSandOn(world, 2,2);

dropSandOn(world, 2,2);

But something different happens any time a cell hits the number “4”...
Part 4: Sandpiles

Here's how dropping sand works:

```java
dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
dropSandOn(world, 2, 2);
```

Boom!
Part 4: Sandpiles

- Here's what happened:
  - When a fourth piece of sand was placed on a square where 3 existed previously, the pile "toppled," meaning the following:
    - It set the value in its current square to 0
Part 4: Sandpiles

Here’s what happened:

- When a fourth piece of sand was placed on a square where 3 existed previously, the pile “toppled,” meaning the following:
  - It set the value in its current square to 0
  - It increased the value in squares in each of the cardinal directions by 1, effectively repeating the “dropping” process in those 4 locations!
- Yes, this means that dropping sand in one location could cause a chain reaction if neighboring (cardinal) cells already have 3 piles of sand in them!
Part 4: Sandpiles

- A few more details about this problem:
  - This function must be done recursively. The sandpile routine should look quite self-similar when you need to topple piles of 4!
Part 4: Sandpiles

- A few more details about this problem:
  - This function must be done recursively. The sandpile routine should look quite self-similar when you need to topple piles of 4!
  - Beware of the bounds of the grid. Any pile that topples on the boundary of a grid should not attempt to modify out of bounds locations.
    - You can determine whether a coordinate set is in bounds with the `grid.inBounds(row, col)` function.
Part 4: Sandpiles

- A few more details about this problem:
  - This function must be done **recursively**. The sandpile routine should look quite self-similar when you need to topple piles of 4!
  - Beware of the bounds of the grid. Any pile that topples on the boundary of a grid should not attempt to modify out of bounds locations.
    - You can determine whether a coordinate set is in bounds with the `grid.inBounds(row, col)` function.
  - In case you need a refresher, here’s how you might access / set the {0,0} element of some existing grid `world`!

```cpp
world[0][0] = 137;

cout << world[0][0] << endl; // 137
```
Questions about Sandpiles?

- Anakin Try(-catch)walker, 106B cynic and disgraced Jedi (in that order)
Part 5: Plotter

- It’s time for the final part of the assignment, and guess what?
Part 5: Plotter

- It’s time for the final part of the assignment, and guess what? You don’t need to use recursion here. We're serious.
Part 5: Plotter

- It’s time for the final part of the assignment, and guess what? You don’t need to use recursion here. We’re serious.
Part 5: Plotter

- You will, however, need to implement a plotter.
Part 5: Plotter

- You will, however, need to implement a plotter.
  - A plotter is a coordinate-based drawing system that turns a series of commands into a drawing on a simply canvas!
Part 5: Plotter

- You will, however, need to implement a plotter.
  - A plotter is a coordinate-based drawing system that turns a series of commands into a drawing on a simply canvas!
  - Here’s an example:

```
MoveAbs -0.8 0.8
PenDown
MoveAbs 0.8 0.8
MoveAbs -0.8 -0.8
MoveAbs 0.8 -0.8
```

Notice that the pen starts at (0,0), so `MoveAbs (-0.8, 0.8)` would move the pen to the upper-left.
Part 5: Plotter

- Here are a few logistics about the plotter:
  - The pen initially begins at (0,0), which in the center of the canvas. It begins **not on the canvas**, with width=1 and color=black.
Part 5: Plotter

- Here are a few logistics about the plotter:
  - The pen initially begins at (0,0), which in the center of the canvas. It begins not on the canvas, with width=1 and color=black.
  - We've provided you the following function:

```java
void drawLine(double x0, double y0,
              double x1, double y1,
              PenStyle info);
```
Part 5: Plotter

- Here are a few logistics about the plotter:
  - The pen initially begins at (0,0), which in the center of the canvas. It begins **not on the canvas**, with width=1
    and color=black
  - We’ve provided you the following function:

```c
void drawLine(double x0, double y0,
               double x1, double y1,
               PenStyle info);
```

I draw a line from (x0,y0) to (x1, y1),
with a pen width and color
determined by the PenStyle struct!
Part 5: Plotter

- Here are a few logistics about the plotter:
  - The pen initially begins at (0,0), which in the center of the canvas. It begins not on the canvas, with width=1 and color=black
  - We've provided you the following function:

```c
void drawLine(double x0, double y0, double x1, double y1, PenStyle info);
```

I draw a line from (x0, y0) to (x1, y1), with a pen width and color determined by the PenStyle struct!

```c
struct PenStyle {
    double width;
    std::string color;
};
```

A struct is a custom-defined container. PenStyle is defined in "plotter.h"
Part 5: Plotter

Here are a few logistics about the plotter:

- The pen initially begins at (0,0), which in the center of the canvas. It begins not on the canvas, with width=1 and color=black.
- We've provided you the following function:

  ```
  void drawLine(double x0, double y0,
                 double x1, double y1,
                 PenStyle info);
  ```

  I draw a line from (x0,y0) to (x1, y1), with a pen width and color determined by the `PenStyle` struct!

PenStyle style;
style.width = 2.71828;
style.color = "orange";
cout << style.color << endl;

Here's how to declare and store properties inside a struct. Remember that width and color were defined by the programmer in the struct signature on the right!

```
struct PenStyle {
  double width;
  std::string color;
};
```

A struct is a custom-defined container. `PenStyle` is defined in "plotter.h"
Part 5: Plotter

Here are a few logistics about the plotter:

○ The pen initially begins at (0,0), which in the center of the canvas. It begins not on the canvas, with width=1 and color=black
○ You will need to implement the following function:

```cpp
void runPlotterScript(istream& script);
```

I contain a series of commands (stored as strings) that you will read in line by line! These commands are instructions for your plotter!
Part 5: Plotter

Here are a few logistics about the plotter:

- The pen initially begins at (0,0), which is the center of the canvas. It begins not on the canvas, with width=1 and color=black.
- You will need to implement the following function:

```cpp
void runPlotterScript(istream& script);
```

I contain a series of commands (stored as strings) that you will read in line by line! These commands are instructions for your plotter.

Here's how you might read individual lines from the `script` stream:

```cpp
for (string line; getline(name-of-stream, line); ) {
    /* ... do something with line ... */
}
```
Part 5: Plotter

- Here are the kinds of commands (lines) that you'll encounter!

**PenDown**: Lowers the pen. Has no effect if the pen is already down.

**PenUp**: Raises the pen. Has no effect if the pen is already up.

**MoveAbs \( x, y \)**: Moves the pen to position \((x, y)\), drawing a line if the pen was down. The values of \(x\) and \(y\) can be any real numbers and might not fit within the bounds of the paper. If that's the case, just move the pen off the page. Our `drawLine` function is smart enough to only draw the part of the line that happens to be on the page.

**MoveRel \( dx, dy \)**: Adds \(dx\) to the pen's \(x\) coordinate and \(dy\) to the pen's \(y\) coordinate, drawing a line if the pen was down. As above, \(dx\) and \(dy\) can be any real numbers and might take the pen off the page.

**PenColor **\( color \)**: Changes the color of the pen to the string given by color. (The `drawLine` function supports basic color names like `red`, `orange`, `yellow`, etc., along with HTML colors like `#41e3a`. If you haven't seen HTML colors before, check out this link for some examples.)

**PenWidth **\( width \)**: Changes the width of the pen to the number given by `width`, which can be any positive real number. Larger widths draw thicker lines.
Part 5: Plotter

- Here are the kinds of commands (lines) that you’ll encounter!

**PenDown**: Lowers the pen. Has no effect if the pen is already down.

**PenUp**: Raises the pen. Has no effect if the pen is already up.

**MoveAbs x y**: Moves the pen to position \((x, y)\), drawing a line if the pen was down. The values of \(x\) and \(y\) can be any real numbers and might not fit within the bounds of the paper. If that’s the case, just move the pen off the page. Our `drawLine` function is smart enough to only draw the part of the line that happens to be on the page.

**MoveRel dx dy**: Adds \(dx\) to the pen’s \(x\) coordinate and \(dy\) to the pen’s \(y\) coordinate, drawing a line if the pen was down. As above, \(dx\) and \(dy\) can be any real numbers and might take the pen off the page.

**PenColor color**: Changes the color of the pen to the string given by `color`. (The `drawLine` function supports basic color names like `red`, `orange`, `yellow`, etc., along with HTML colors like `#c41e3a`. If you haven’t seen HTML colors before, check out this link for some examples.)

**PenWidth width**: Changes the width of the pen to the number given by `width`, which can be any positive real number. Larger widths draw thicker lines.

Some things I want you to consider:
- There’s currently no provided way to determine whether the pen is **up** or **down**. You’ll need to figure out how to keep track of that!
Part 5: Plotter

Here are the kinds of commands (lines) that you’ll encounter!

**PenDown**: Lowers the pen. Has no effect if the pen is already down.

**PenUp**: Raises the pen. Has no effect if the pen is already up.

**MoveAbs** $x\ y$: Moves the pen to position $(x, y)$, drawing a line if the pen was down. The values of $x$ and $y$ can be any real numbers and might not fit within the bounds of the paper. If that’s the case, just move the pen off the page. Our `drawLine` function is smart enough to only draw the part of the line that happens to be on the page.

**MoveRel** $dx\ dy$: Adds $dx$ to the pen’s $x$ coordinate and $dy$ to the pen’s $y$ coordinate, drawing a line if the pen was down. As above, $dx$ and $dy$ can be any real numbers and might take the pen off the page.

**PenColor** $color$: Changes the color of the pen to the string given by $color$. (The `drawLine` function supports basic color names like red, orange, yellow, etc., along with HTML colors like #c41e3a. If you haven’t seen HTML colors before, check out this link for some examples.)

**PenWidth** $width$: Changes the width of the pen to the number given by $width$, which can be any positive real number. Larger widths draw thicker lines.

Some things I want you to consider:

- There’s currently no provided way to determine whether the pen is **up** or **down**. You’ll need to figure out how to keep track of that!
- The above is true about your pen’s current coordinates as well! Sounds like you’ll have some extra bookkeeping to do…
Part 5: Plotter

- Here are the kinds of commands (lines) that you'll encounter!

**PenDown:** Lowers the pen. Has no effect if the pen is already down.

**PenUp:** Raises the pen. Has no effect if the pen is already up.

**MoveAbs** $x \ y$: Moves the pen to position $(x, y)$, drawing a line if the pen was down. The values of $x$ and $y$ can be any real numbers and might not fit within the bounds of the paper. If that's the case, just move the pen off the page. Our `drawLine` function is smart enough to only draw the part of the line that happens to be on the page.

**MoveRel** $dx \ dy$: Adds $dx$ to the pen’s $x$ coordinate and $dy$ to the pen’s $y$ coordinate, drawing a line if the pen was down. As above, $dx$ and $dy$ can be any real numbers and might take the pen off the page.

**PenColor** `color`: Changes the color of the pen to the string given by `color`. (The `drawLine` function supports basic color names like red, orange, yellow, etc., along with HTML colors like #c41e3a. If you haven’t seen HTML colors before, check out this link for some examples.)

**PenWidth** `width`: Changes the width of the pen to the number given by `width`, which can be any positive real number. Larger widths draw thicker lines.

Some things I want you to consider:

- There’s currently no provided way to determine whether the pen is up or down. You’ll need to figure out how to keep track of that!

- The above is true about your pen’s current coordinates as well! Sounds like you’ll have some extra bookkeeping to do...

- The commands are case insensitive. You should be able to interpret the command: “penColOr BLue”
Part 5: Plotter

- Here are the kinds of commands (lines) that you'll encounter!

**PenDown**: Lowers the pen. Has no effect if the pen is already down.

**PenUp**: Raises the pen. Has no effect if the pen is already up.

**MoveAbs x y**: Moves the pen to position \((x, y)\), drawing a line if the pen was down. The values of \(x\) and \(y\) can be any real numbers and might not fit within the bounds of the paper. If that’s the case, just move the pen off the page. Our `drawLine` function is smart enough to only draw the part of the line that happens to be on the page.

**MoveRel dx dy**: Adds \(dx\) to the pen’s \(x\) coordinate and \(dy\) to the pen’s \(y\) coordinate, drawing a line if the pen was down. As above, \(dx\) and \(dy\) can be any real numbers and might take the pen off the page.

**PenColor color**: Changes the color of the pen to the string given by `color`. (The `drawLine` function supports basic color names like `red`, `orange`, `yellow`, etc., along with HTML colors like `#c41e3a`. If you haven’t seen HTML colors before, check out this link for some examples.)

**PenWidth width**: Changes the width of the pen to the number given by `width`, which can be any positive real number. Larger widths draw thicker lines.

Some things I want you to consider:

- There’s currently no provided way to determine whether the pen is **up** or **down**. You’ll need to figure out how to keep track of that!

- The above is true about your pen’s current coordinates as well! Sounds like you’ll have some extra bookkeeping to do...

- The commands are **case insensitive**. You should be able to interpret the command: “penColOr BLue”

I’m gonna post on this slide for a second for questions!
Part 5: Plotter

- Here's another example list of commands:

```
MoveAbs 0 1
PenDown
MoveRel 0 -1.2
PenUp
MoveRel 0 -0.3
PenDown
MoveAbs 0 -1
```
Part 5: Plotter

- Here’s another example list of commands:

  MoveAbs 0 1
  PenDown
  MoveRel 0 -1.2
  PenUp
  MoveRel 0 -0.3
  PenDown
  MoveAbs 0 -1

- Notice that many of the commands include numbers as arguments.
Part 5: Plotter

- Here's another example list of commands:

  MoveAbs 0 1
  PenDown
  MoveRel 0 -1.2
  PenUp
  MoveRel 0 -0.3
  PenDown
  MoveAbs 0 -1

- Notice that many of the commands include numbers as arguments.
- These arguments will always be delimited (separated) by spaces in a line, so if you want to turn a single command into a vector of tokens (items separated by delimiters), use the following function: (in "strlib.h")

  Vector<string> name-of-result = stringSplit(string-to-split, what-to-split-on);
Part 5: Plotter

- Here’s another example list of commands:

```
MoveAbs 0 1
PenDown
MoveRel 0 -1.2
PenUp
MoveRel 0 -0.3
PenDown
MoveAbs 0 -1
```

- Notice that many of the commands include numbers as arguments.
- These arguments will always be delimited (separated) by spaces in a line, so if you want to turn a single command into a vector of tokens (items separated by delimiters), use the following function: (in “strlib.h”)

```
Vector<string> name-of-result = stringSplit(string-to-split, what-to-split-on);
```
- You can turn string representations of numbers into floating point numbers via `stringToReal(string s)`
Part 5: Plotter

- Some final thoughts:
  - You can assume that the stream provided to you is properly formatted -- even though the case might be strange, data will always be formatted correctly like it was in the examples :)

Part 5: Plotter

- Some final thoughts:
  - You can assume that the stream provided to you is properly formatted -- even though the case might be strange, data will always be formatted correctly like it was in the examples :)
  - You don’t need to worry about out of bounds errors, just follow whatever coordinates we give you :). Oh happy day!
Part 5: Plotter

- Some final thoughts:
  - You can assume that the stream provided to you is properly formatted -- even though the case might be strange, data will always be formatted correctly like it was in the examples :)
  - You don't need to worry about out of bounds errors, just follow whatever coordinates we give you :). Oh happy day!
  - Remember to initialize the plotter (set the initial position to (0,0), and set the color=black and width=1) before you begin reading from the stream!
Part 5: Plotter

- Some final thoughts:
  - You can assume that the stream provided to you is properly formatted -- even though the case might be strange, data will always be formatted correctly like it was in the examples :)
  - You don't need to worry about out of bounds errors, just follow whatever coordinates we give you :). Oh happy day!
  - Remember to initialize the plotter (set the initial position to (0,0), and set the color=black and width=1) before you begin reading from the stream!
  - Whenever dealing with strings, take another looksie through "strlib.h"! You might surprise yourself with the helpful functions!
Part 5: Plotter

- Some final thoughts:
  - You can assume that the stream provided to you is properly formatted -- even though the case might be strange, data will always be formatted correctly like it was in the examples :)
  - You don't need to worry about out of bounds errors, just follow whatever coordinates we give you :). Oh happy day!
  - Remember to initialize the plotter (set the initial position to (0,0), and set the color=black and width=1) before you begin reading from the stream!
  - Whenever dealing with strings, take another looksie through "strlib.h"! You might surprise yourself with the helpful functions!
  - You'll be testing your plotter by running pre-packaged plotter scripts, and then you will manually inspect them for correctness. Don't worry -- if your implementation is incorrect you should be able to tell by inspection :)
Questions about Plotter?

An old Hewlett-Packard Plotter machine. Kinda... ugly, no?
Congrats! You’re ready to tackle A1!

Good luck! If you get stuck, remember that you have your wonderful Section Leaders and LAIR hours for help!