Data structures

Ch ???
Multi-dimension arrays

2D arrays have a row and column index, however this is a bit misleading

Computers actually only have a 1D memory...

We are just pretending like there is more...

How can we do this?
Multi-dimension arrays

Same way we have 2D maps: make some assumptions then project
Multi-dimension arrays

A 2D matrix is split up by rows, for example:

```java
int x[3][5];
```

We think of this as:

- x0,0  x0,1  x0,2  x0,3  x0,4
- x1,0  x1,1  x1,2  x1,3  x1,4
- x2,0  x2,1  x2,2  x2,3  x2,4

But the computer sees:

- x0,0  x0,1  x0,2  x0,3  x0,4  x1,0  x1,1  x1,2...
Multi-dimension arrays

So even if we declare x as:

```c
int x[3][5];
```

We can access it by either:

```c
x[1][4]
```

or

```c
((int*)x)[1*5 + 4]
```

(see: arrayCheat.cpp)
Arrays

What are some properties of arrays?
Pros:

Cons:
What are some properties of arrays?
Pros:
1. Instantly access any spot
2. Built into many languages (C++ too!)

Cons:
1. Hard to add to end (sorta can)
2. Fixed size/length
3. Inserting in the middle is very annoying
Arrays

Partially filled arrays can get around inserting at end issue

Dynamically created arrays can get around fixed size issue in conjunction with above

(if out of space, ask for new array twice as big and copy over old stuff)
Arrays

We can never really fix insertion though...

The biggest problem is inserting depends on how many other things are already in the array

So big arrays inserting is much much much harder!
Linked List

We get around this problem with pointers

Instead of using an array directly, we will make a mini-class with a single int and a pointer to another type as itself

class “item”

(see: linkedList.cpp)
To insert an item into a linked list, we simply need to make a new box, then change where the previous and new box are pointing.
Linked List

To add to the end of a linked list, we first need to go to the end, then simply add a new box here and change the last link.

To find the last item, we keep checking “next” then moving to it, until “next” is nullptr (a pointer to nowhere/the abyss)
Linked lists have very easy inserts (you don't need to shift anything, thanks to pointers!)

However, access time is not fast...

If you want the last item, you would need to go through all others (one at a time)

Normal array: slow insert, fast access
Linked list: fast insert, slow access
I have mentioned a stack a few times before...

This is how function calls work, and they are a specific type of linked list, but with only two simple actions

1. Push (add new item to “top” of stack)
2. Pop (take top item off stack)
Stack

Suppose we have this stack (pancake... yum!):

In this case if we “push”, we flip another pancake on top
Suppose we have this stack (pancake... yum!):

In this case if we “push”, we flip another pancake on top
Suppose we pushed a few times to get this:

Then a “pop” would remove the top pancake (most recent)
Stack

Suppose we pushed a few times to get this:

Then a “pop” would remove the top pancake (most recent)
“Pushing” is similar to inserting in linked list:
(Step 0. Make new box)
Step 1. Point new box to old top (next box)
Step 2. Change top to point a new box
“Popping” can be done by simply changing the “top” to the one below (but memory leak)

The proper way is:
Step 1. Save old top (so you don't lose it)
Step 2. Change top to one below
Step 3. Delete top (see: stack.cpp)