# Data Structures

# C++ Review

CS 225

Brad Solomon

August 26, 2026

Suggest Music On Discord #music-suggestions

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

# Do you want to do research?. . .

## . . . Are you a freshman or sophomore?

## Come apply to **URSA**!

**U**ndergraduate **R**esearch in **S**cientific **A**dvancement

### Benefits:

- ☑ Research Experience
- ☑ Networking
- ☑ Soft and Hard Skill Development
- ☑ 1 credit hour + GPA boost
- ☑ Resume Booster

student run

apply for fall

**Scan for:**

- Website
- Application
- Interest form

# Theta Tau

*The Nation's Largest, Oldest, and Foremost Co-Ed Professional Engineering Fraternity*

## Brotherhood

- Lifelong Friendships
- Supportive Community through collegiate & post-grad career
- Retreats, Camping, Intramurals and more!

## Professionalism

- Mentorship from upperclassmen
- Access to vast Theta Tau Alumni Network
- Resume Building & Interview Prep
- Skill Building Workshop

## Service

- National Service Initiatives (Habitat for Humanity)
- Community Service Projects
- Volunteering
- Fundraising

Register for Rush!

**Info Session 1**:
9/1 @ 6:30 PM CIF 3039

**Info Session 2**:
9/8 @ 5:30 PM CIF 3039

# Add suggested music (on Discord)

#music-suggestions channel

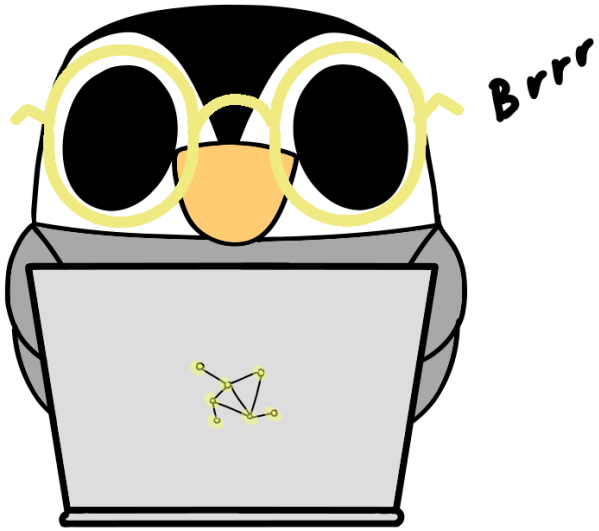
# (Optional) Open Lab This Week

This week's lab is open office hours

Focus is making sure your machine is setup for semester

Installation information available on website

(See 'Comprehensive Setup Guide')

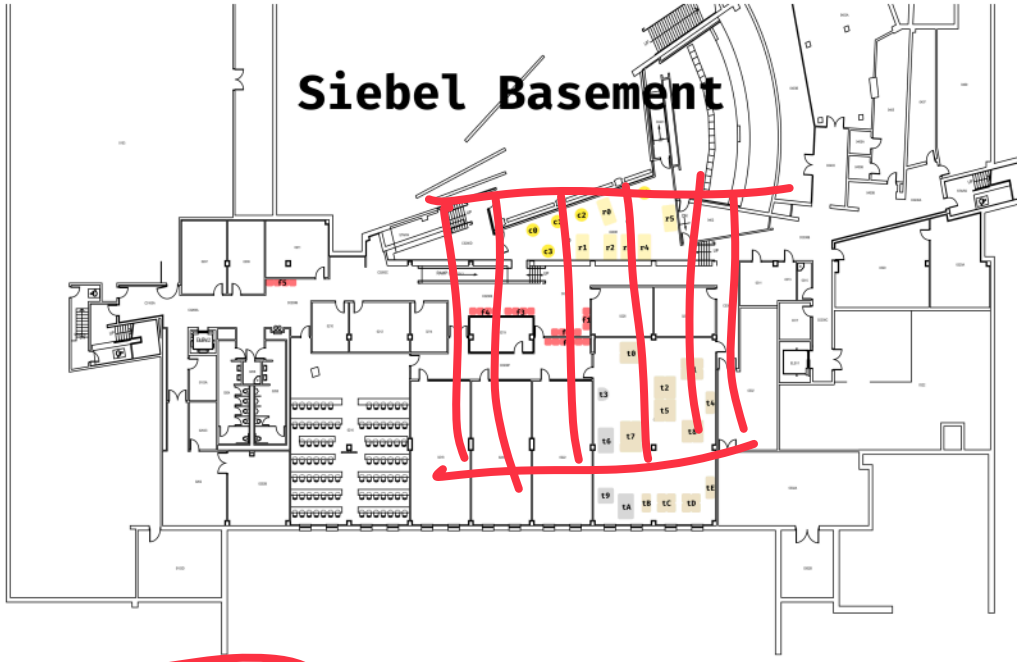


# Office Hour Etiquette

Schedule and link to queue on the website

Pay attention to the rules!

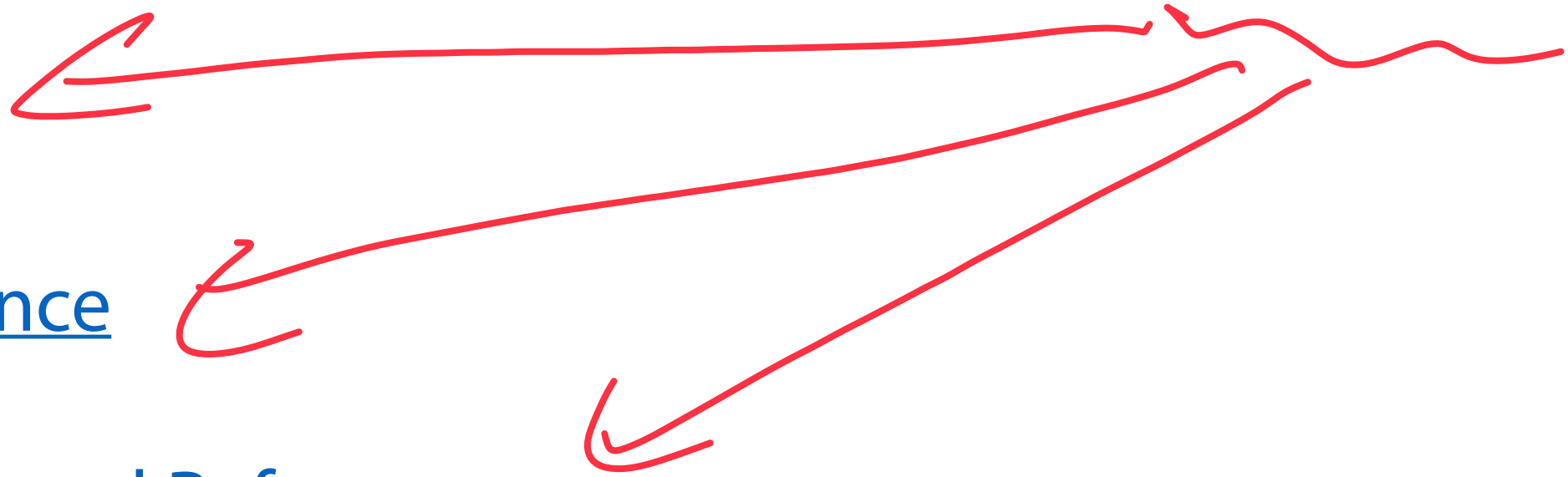
1. Be in Siebel Basement
2. Tag questions
3. Ask **one** specific question
4. Include a specific location
5. Include both your name and Discord ID



# New Resources on Website

The Resources page on websites updates with lectures!

Also has resources you can see at any time (hit ‘see all’):



Classes

Inheritance

Pointers and References

# Testing a 'Clicker' Set-up!

Have you signed up to take exam 0?

A) Yes! 83%

B) No! 17%

C) I'm still on the waitlist… 1%

Join Code: 225

You can participate by going to website:

https://clicker.cs.illinois.edu/

# Exam 0 (9/2 — 9/4)

An introduction to CBTF exam environment / expectations

Quiz on foundational knowledge from all pre-reqs

Practice questions can be found on PL

Topics covered can be found on website

**Registration open now :)**

https://courses.engr.illinois.edu/cs225/exams/

# Learning Objectives

A brief high level review of C++

- Fundamentals of Objects / Classes
- Pointers
- Memory Management and Ownership

mp- puzzle

Brainstorm the List Abstract Data Types (ADT)

# Encapsulation - Classes

Abstraction / organization separating:

**Internal Implementation**

↳ How the code does

↳ One piece at a time

↳ Test your work!

**External Interface**

↳ what the code is doing

↳ website problem page

↳ Doxygen

Tip: Always Start here!

↳ Draw it out!

# Brainstorming a 'Library' class

header file ≡ interface

```
1  class Library {
2  public:
3
4
5
6
7
8
9
10
11
12
13 private:
14
15
16
17
18
19
20
21 };
```

↳ Constructor

↳ Get / Set Book

↳ CheckOut / Add new

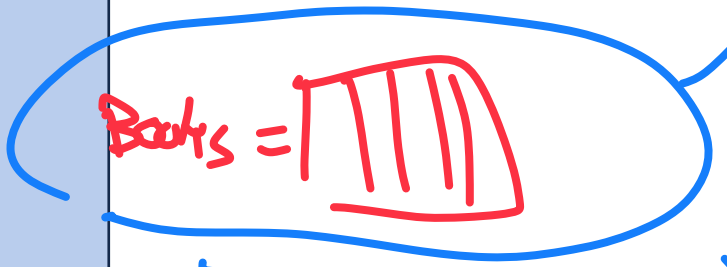
↳ Modify library itself

} General library access rules

↳ Books

↳ Address of library

↳ Employees

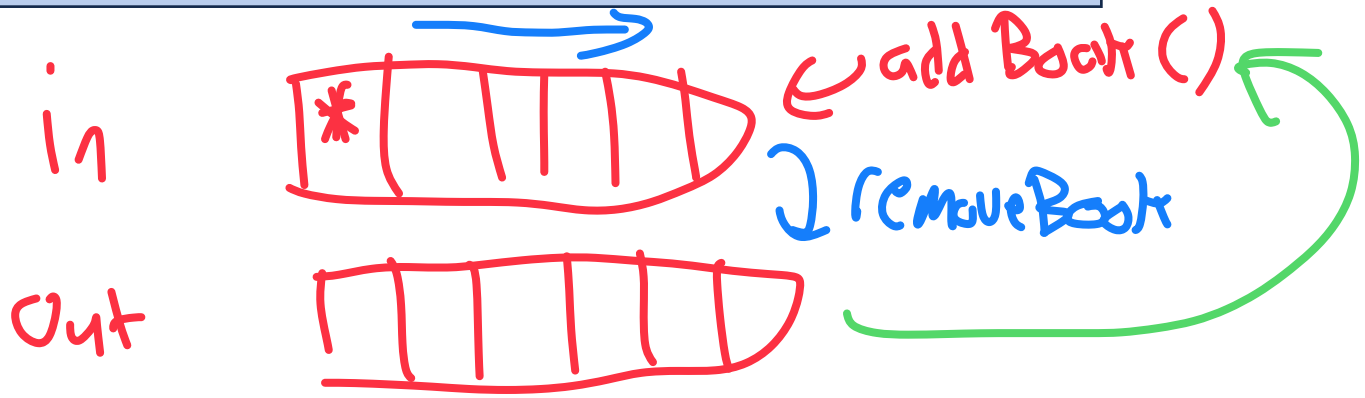



If Brad was writing implementation

# Memory Management — Ownership

## Imagine I have a Library class (and hidden Book class):

```
 1 class Library{
 2 public:
 3     void addBook(Book * book);
 4     void removeBook(std::string title);
 5     void returnBook(Book * book);
 6
 7 private:
 8     std::vector<Book*> in;
 9     std::vector<Book*> out;
10 };
11
```



# Memory Management — Ownership

Imagine I have a Library class:

```
1  class Library{
2  public:
3      void addBook(Book * book);
4      void removeBook(std::string title);
5      void returnBook(Book * book);
6      # Probably some
7  private:
8      std::vector<Book*> in;
9      std::vector<Book*> out;
10 };
11
```

Join Code: 225

**Pretest:** Does Library class 'own' the Books it is storing?

A) **Yes!** 17%

B) **No!** 75%

C) **Not sure** ~8%

# Pointers

Pointers store memory addresses

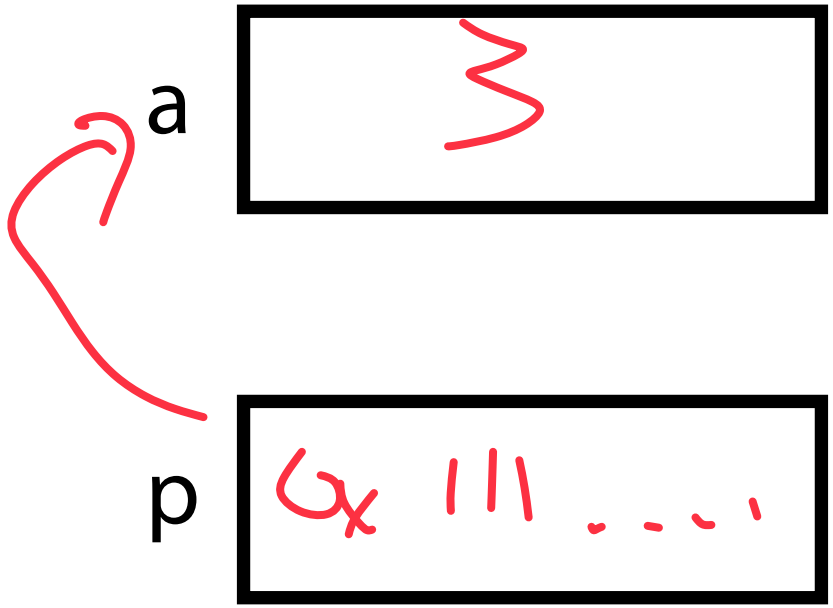
```
int a = 3;
```

stack

```
int *p = &a;
```

pointer (also on stack)

address of (a)

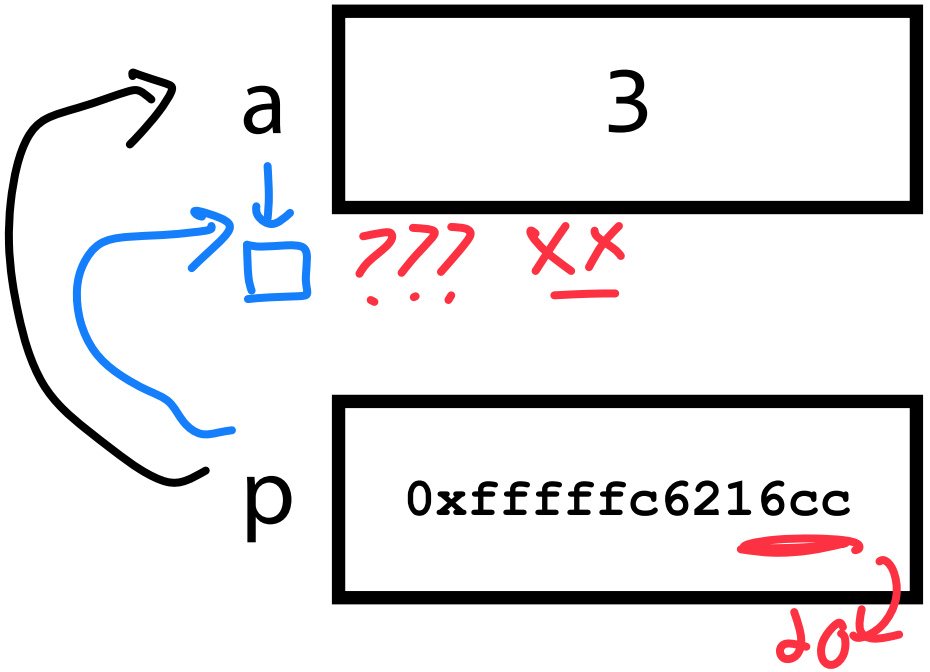


# Pointers

Pointers store memory addresses

```
int a = 3;

int *p = &a;

p++;
```

of type int (4 bytes)




Does a change? Does p?

↳ No

↳ Yes

# Pointers

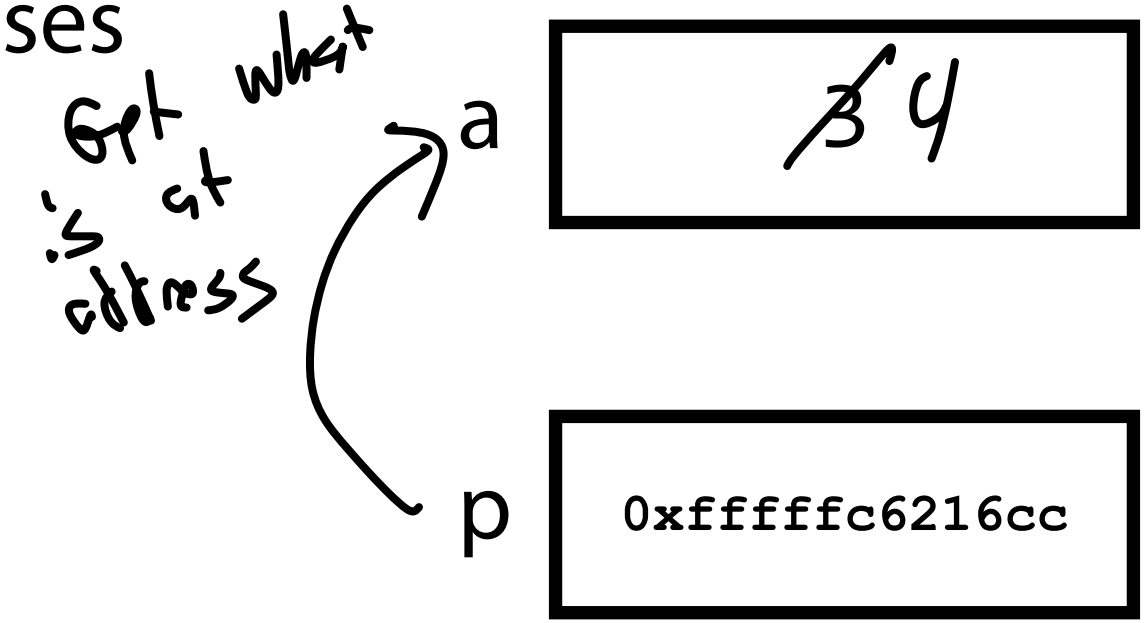
Pointers store memory addresses

```
int a = 3;

int *p = &a;

(*p)++;
```

↑ dereference operator

Does a change? Does p?

↳ Yes ↳ No!



# Memory Management

**Stack**: Local variable storage

**Ex:** `int x = 5;`

local scope of function

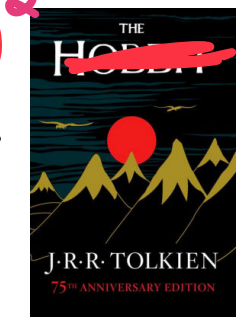
**Heap:** Dynamic storage

**Ex:** `int* x = new int[5];`

If we make it we must delete it ourselves

Ownership over things we make

# Memory Management - Parameters




Pass by **Value:** A local copy of the original

Ex: addBook(Book book)

Pass by **Pointer to Value:** An address on the heap

Ex: addBook(Book* book)

Pass by **Reference:** An *alias* to an existing variable

Ex: addBook(Book& book)

# Memory Management - Parameters

## Which implementation do you prefer?

```
1  class Library {
2  public:
3      int numBooks;
4      std::string * titles;
5  };
6
7  // *** Function A ***
8  std::string getFirstBook(Library l){
9      return (l.numBooks > 0) ? l.titles[0] : "None";
10 }
11
12 // *** Function B ***
13 std::string getFirstBook(Library * l){
14     return(l->numBooks > 0) ? l->titles[0] : "None";
15 }
16
17 // *** Function C ***
18 std::string getFirstBook(Library & l){
19     return (l.numBooks > 0) ? l.titles[0] : "None";
20 }
21
22
23
24
```

Skip!

# Memory Management

Local memory on the stack is managed by the computer

Heap memory allocated by **new** and freed by **delete**

Pass by value makes a copy of the object

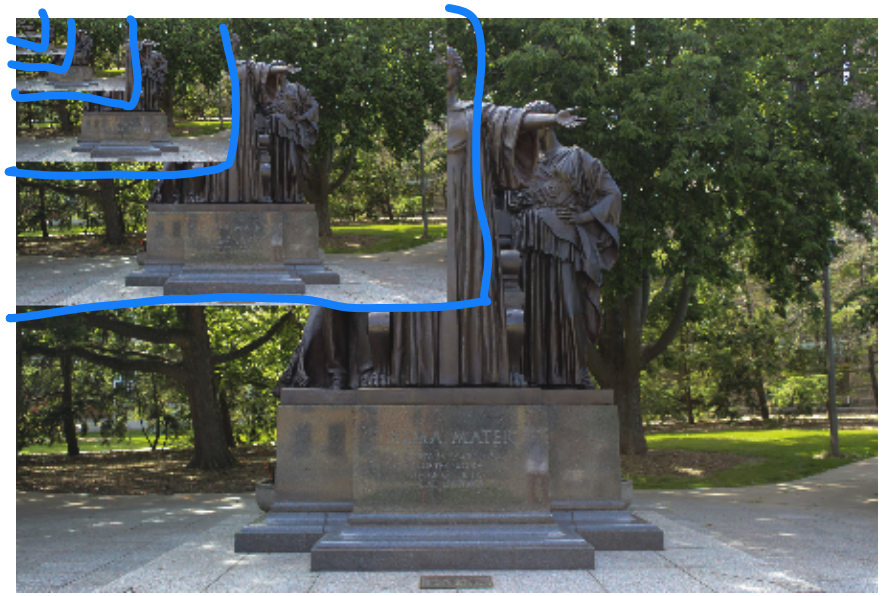
Pass by pointer can be dereferenced to modify an object

Pass by reference modifies the object directly

# Memory Management — Ownership

What does **ownership** mean in C++?

↳ who created it and who needs to free it



↑ - Creating 6 copies
- one image that reuse

Interface says create
stacked image of resized
Master image

Image
PNG
Sticker sheet

# Memory Management — Ownership

```
 1 class Library{
 2 public:
 3     void addBook(Book * book);
 4
 5
 6     void removeBook(std::string title);
 7
 8
 9     void returnBook(Book * book);
10 private:
11
12     std::vector<Book*> in;
13
14
15     std::vector<Book*> out;
16
17
18 };
```

Does Library 'own' Books?

A) **Yes!** 25%

B) **No!** 72%

C) **Not sure** 3%

# Memory Management — Ownership

```
1  class Library{
2  public:
3      void addBook(Book * book);
4
5
6      void removeBook(std::string title);
7
8
9      void returnBook(Book * book);
10 private:
11
12     std::vector<Book*> in;
13
14
15     std::vector<Book*> out;
16
17
18 };
```

Could be more than one library!

Does Library 'own' Books?

A) **Yes!**

B) **No!**

C) **Not sure**

Are they destroyed when the Library destructor is called?

↳ Books not allocated by library shouldn't be deleted by library

# Memory Management — Ownership

```
 1 class Library{
 2 public:
 3     void addBook(Book book);
 4
 5
 6     void removeBook(std::string title);
 7
 8
 9     void returnBook(Book book);
10 private:
11
12     std::vector<Book> in;
13
14
15     std::vector<Book> out;
16
17
18 };
```

Does Library 'own' Books?

A) **Yes!** 80%

B) **No!**

C) **Not sure**

# Memory Management — Ownership

```
1  class Library{
2  public:
3      void addBook(Book book);
4
5
6      void removeBook(std::string title);
7
8
9      void returnBook(Book book);
10 private:
11
12     std::vector<Book> in;
13
14
15     std::vector<Book> out;
16
17
18 };
```

we delete copies only

Does Library 'own' Books?

A) **Yes!**

B) **No!**

C) **Not sure**

↖ we own our own copies of the Books

Are they destroyed when the Library destructor is called?

↳ Indirectly yes!

Library → owned → vector → owned → Book

# Memory Management — Ownership

```
1  class Library{
2  public:
3      void addBook(const Book& book);
4
5
6      void removeBook(std::string title);
7
8
9      void returnBook(const Book& book);
10 private:
11
12     std::vector<Book*> in;
13
14
15     std::vector<Book*> out;
16
17
18 };
```

Does Library 'own' Books?

A) **Yes!**

B) **No!**

C) **Not sure**

Are they destroyed when the Library destructor is called?

# Memory Management — Ownership

**The owner of an object is responsible for its resource management (particularly allocation / deallocation)**

A ‘litmus test’ of ownership — who handles destruction?

If we are storing pointers or references, not our problem!

good heuristic / rule of thumb

Vector’s consolation prize — vector handles destruction

# The Rule of Three

Stopped here

If it is necessary to **define any one** of these three functions in a class, it will be necessary to **define all three** of these functions:

1. Destructor — Called when we delete object

2. Copy Constructor — Make a new object as a copy of an existing one

3. Copy assignment operator — Assign value from existing X to Y

# ‘The Rule of Zero'

## A corollary to Rule of Three

Classes that **declare** custom destructors, copy/move constructors or copy/move assignment operators should deal exclusively with ownership. Other classes **should not declare** custom destructors, copy/move constructors or copy/move assignment operators

— Scott Meyers

```
 1 class Library {
 2 public:
 3     int numBooks;
 4     std::string * titles;
 5     ~Library();
 6     Library( int num, std::string* list );
 7 };
 8
 9 Library::~Library(){
10    delete[] titles;
11    titles = nullptr;
12 }
13
14 Library::Library(int num, std::string* list){
15     numBooks = num;
16     titles = new std::string[ num ];
17     std::copy(inList, inList + num, titles);
18 }
19
20 int main(){
21    std::string myBooks[3] = {"A", "B", "C"};
22    Library L1( 3, myBooks );
23    Library L2( L1 );
24    return 0;
25 }
```

```
1  class Library {
2  public:
3      int numBooks;
4      std::string * titles;
5      ~Library();
6      Library( int num, std::string* list );
7  };
8
9  Library::~Library(){
10    delete[] titles;
11    titles = nullptr;
12 }
13
14 Library::Library(int num, std::string* list){
15     numBooks = num;
16     titles = new std::string[ num ];
17     std::copy(inList, inList + num, titles);
18 }
19
20 int main(){
21    std::string myBooks[3] = {"A", "B", "C"};
22    Library L1( 3, myBooks );
23    Library L2( L1 );
24    return 0;
25 }
```

**Whats wrong with this code?**

A. Can't create L2 Library obj

B. Don't delete either Library

C. The second object being deleted crashes

# Questions?

# Templates

A way to write generic code whose type is determined during completion

# Templates

A way to write generic code whose type is determined during completion


1. Templates are a recipe for code using generic types

# Templates

A way to write generic code whose type is determined during completion


1. Templates are a recipe for code using generic types

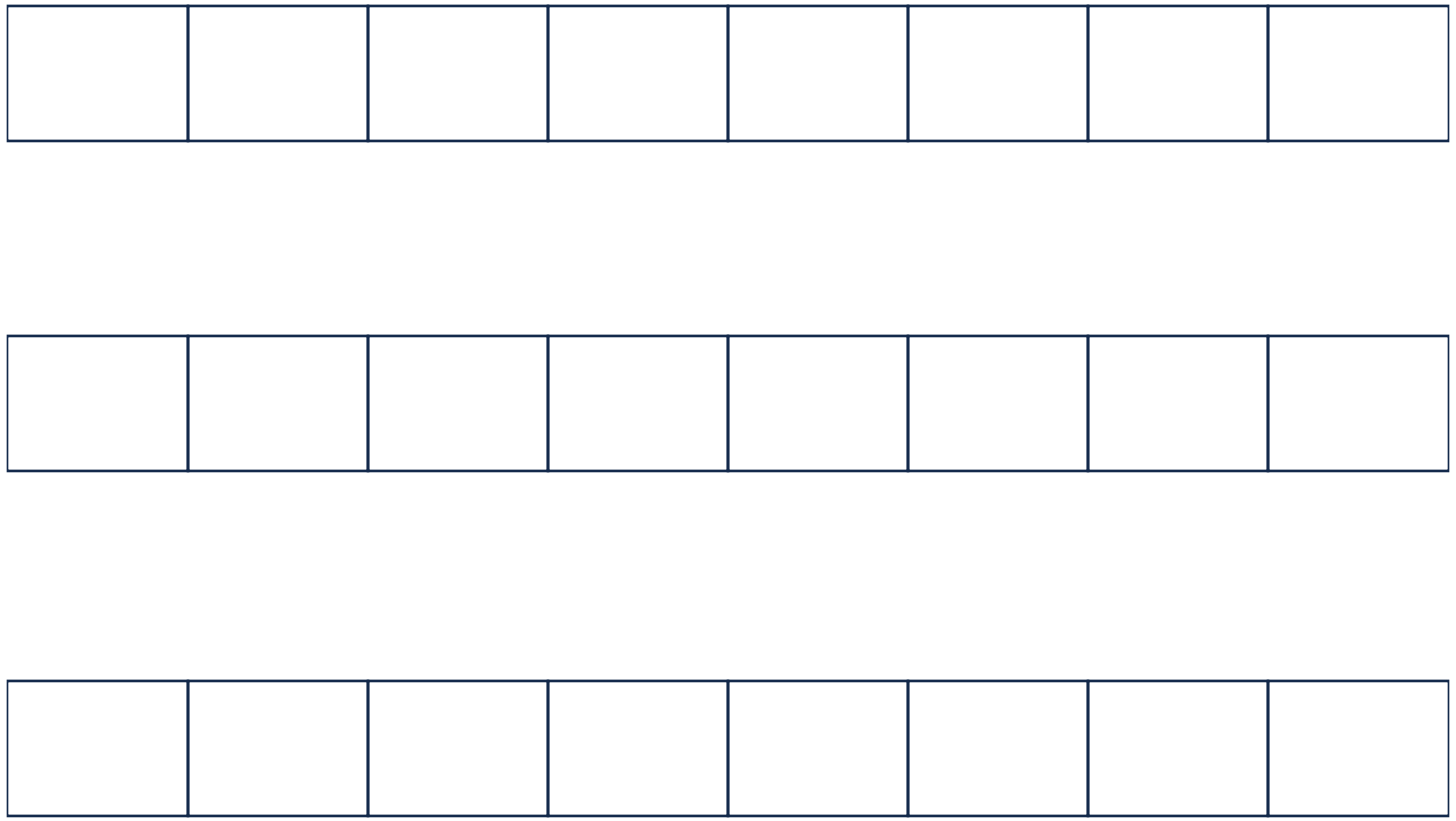
2. The compiler uses templates to generate C++ code **when needed**

```
template <typename T>
T sum(T a, T b){
...
}
```

# template1.cpp

```
1  template <typename T>
2  T max(T a, T b) {
3    T result;
4    result = (a > b) ? a : b;
5    return result;
6  }
7
```

# Templates are very useful!



# List Abstract Data Type

What is the expected **interface** for a list?