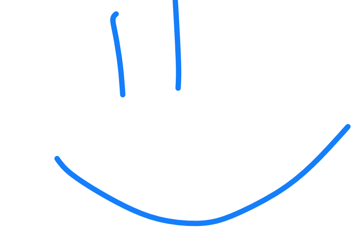
Data Structures
C++ Review

CS 225
Brad Solomon & G Carl Evans

August 23, 2023



Department of Computer Science

Do you want to do research?

Come apply to **PURE**!

Promoting **U**ndergraduate **R**esearch in **E**ngineering

Benefits:

- ☑ Research experience
- ☑ Networking
- ☑ Soft and hard skill development
- ☑ 1 credit hour + GPA boost
- ☑ Resume Booster

Scan for:

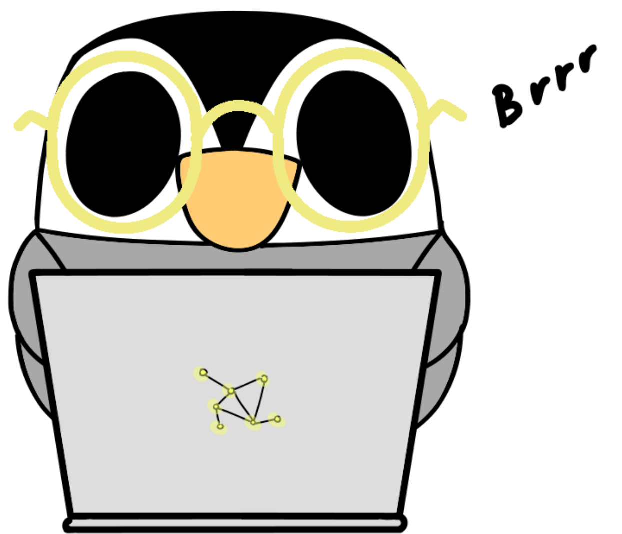
- Interest form
- Website
- Discord

(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



Exam 0 (August 29 — 31)

https://courses.engr.illinois.edu/cs225/fa2023/exams/

An introduction to CBTF exam environment / expectations

Quiz on foundational knowledge from all pre-reqs

Practice questions can be found on PL

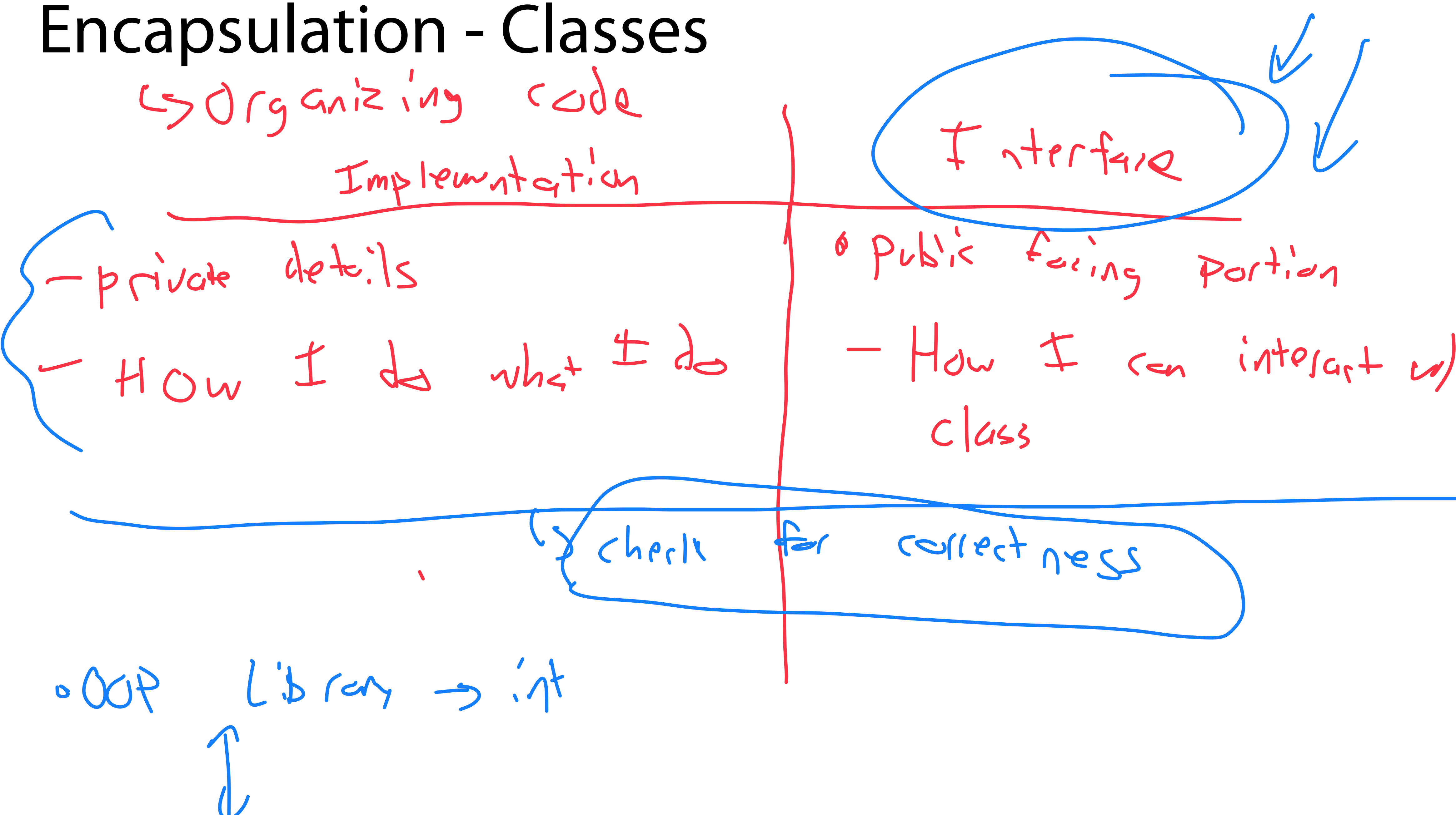
Registration starts August 24

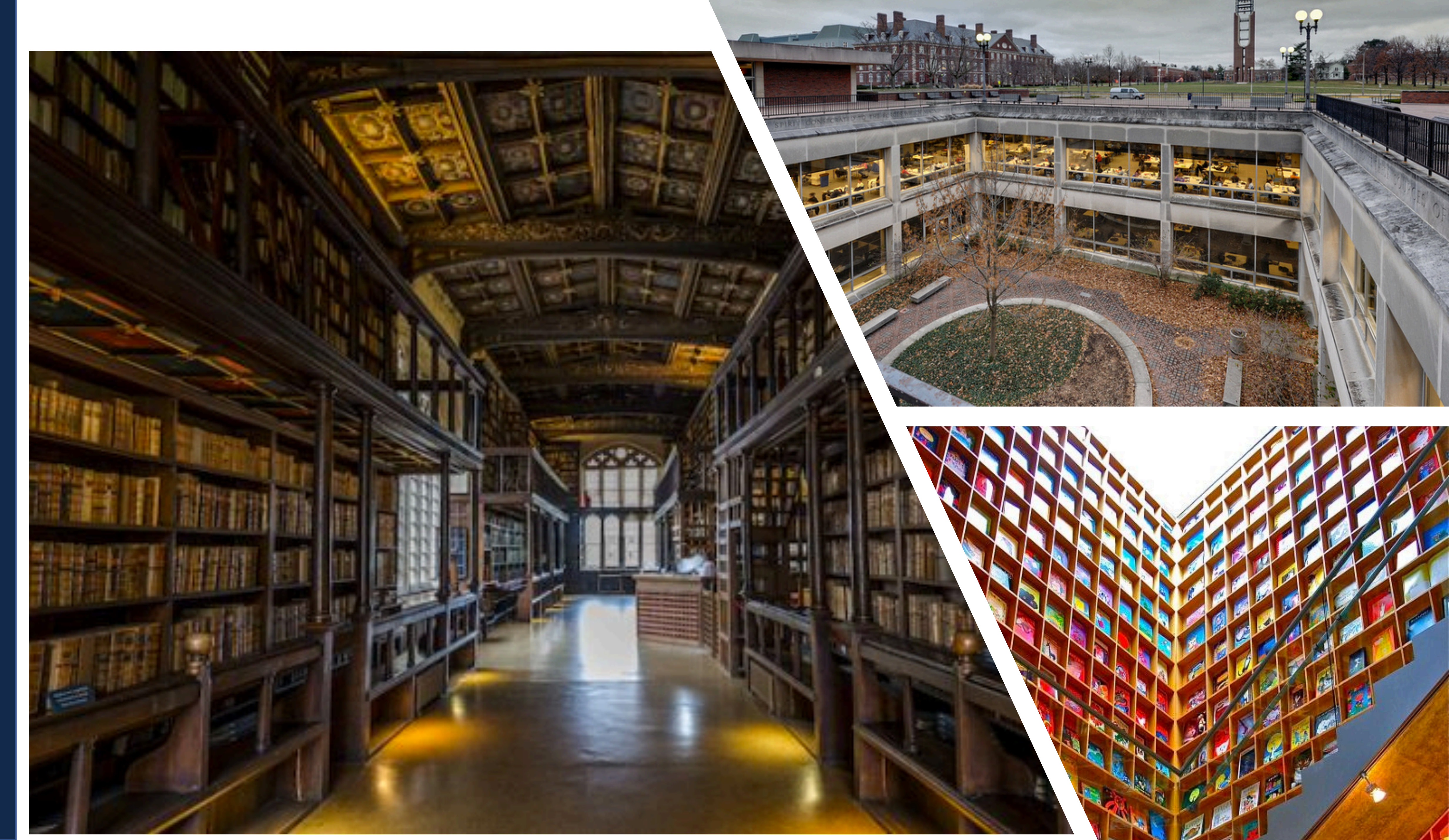
Learning Objectives

A brief high level review of C++

- Fundamentals of Classes
- The Rule of Three
- Memory management
- Function parameters and const
- Templates

Introduce Abstract Data Types (ADT)


Encapsulation - Classes
↳ organizing code
Implementation
Interface
- private details
- How I do what I do
• public facing portion
- How I can interact w/ class
↳ check for correctness
• OOP library → int



Drafting a ‘Library’ class

```
1  class Library {
2  public:
3      checkOutBook()/checkinBook()
4
5
6      bookInLibrary()
7
8
9
10     ~/Close
11
12
13     accessor (get/set)
14
15 private:
16
17 class Bookshelf                users
18
19
20     ↳ Books
21
22
23     Capacity
24
25     delete
};
```

Public (interface)

(implementation)

Class Fundamentals

Constructor – called when you make instance of class

Library( )

Library(Booklist)

(, UserList) . . . .

Destructor – Last thing ever called by class

– Destroyed

Class Fundamentals

Does our library class need a destructor?

Unsure!

The Rule of Three

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
2. Copy Constructor
3. Copy Assignment =

```
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 = inNum;
16     titles = new std::string[ inNum ];
17     std::copy(inList, inList + inNum, 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 }
```

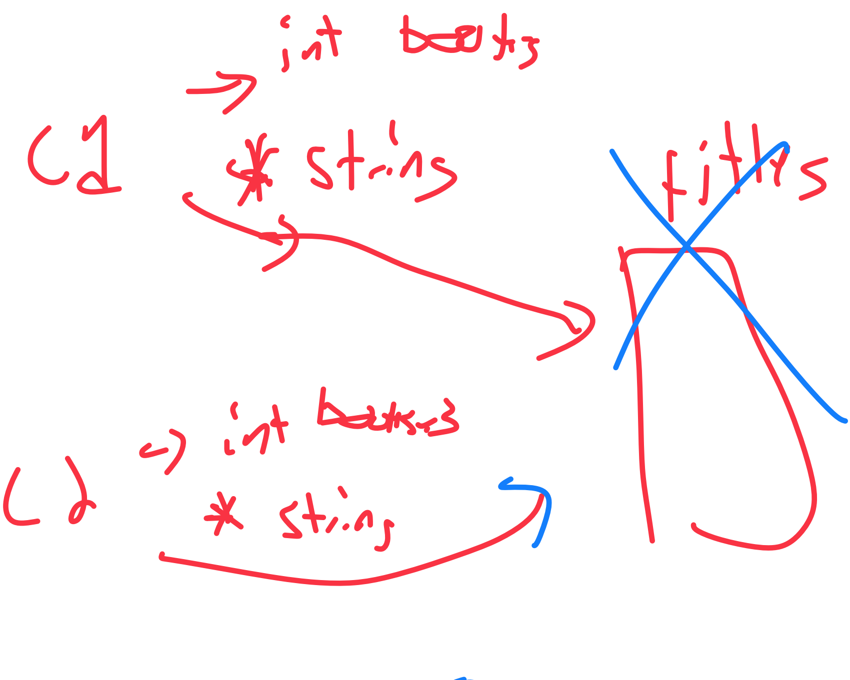
```
1  class Library {
2  public:
3       int numBooks;
4       std::string * titles;
5       ~Library();
6       Library( int num, std::string* list );
7  };
8
9  Library::~Library(){
10     delete titles;
11     titles = nullptr;
12 }
13
14 Library::Library(int num, std::string* list){
15      numBooks = inNum;
16      titles = new std::string[ inNum ];
17      std::copy(inList, inList + inNum, 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. Deleting L1 deletes L2



‘The Rule of Zero'

If you define a destructor, copy, or assignment operator, **you should define all three!**

Implicit default operators are generated otherwise.

Tip: If you can, avoid writing these operators at all!

Memory Management

Stack ☺ Local variable / Managed by the computer ✓
↳ Smaller than you might think (restricted)

Heap dynamic storage / managed by you
↳ new (delete)

Global
int
main()

```
function()
    static int x;
    x++;
```

Reference and Dereference

```
int a = 3;
int b = 5;

int *p = &a;

int &r = b;

cout << p << " " << *p << endl;

cout << r << endl;

p++;
r++;

cout << a << " " << b << endl;

cout << p << " " << *p << endl;

cout << r << endl;
```

Reference (&)

Dereference (*)

Reference and Dereference

Post-class Bonus slide

Students after class were confused about the values here

Blue is the most relevant detail! Red is also important

```
int a = 3;
int b = 5;

int *p = &a; // Value: 0xfffffc6216cc

int &r = b; // Value: 5

cout << p << " " << *p << endl; // Output: "0xfffffc6216cc 3"

cout << r << endl; // Output: "5"

p++; // This increments the POINTER ADDRESS
r++; // This increments b's value

cout << a << " " << b << endl; // "3 6"

cout << p << " " << *p << endl; // "0xfffffc6216d0 -60680480"

cout << r << endl; // Output: "6"
```

This is a pointer value

We can accidentally increment it like any other value

It now points to something I dont own!

Memory Management - Parameters

Value

Value — Pointer

Reference

Memory Management - Parameters

```
1  class Library {
2  public:
3      int numBooks;
4      std::string * titles;
5
6  };
7
8  // *** Function A ***
9  std::string getFirstBook(Library l){
10     return (l.numBooks > 0) ? l.titles[0] : "None";
11 }
12
13
14 // *** Function B ***
15 std::string getFirstBook(Library * l){
16     return(l->numBooks > 0) ? l->titles[0] : "None";
17 }
18
19
20 // *** Function C ***
21 std::string getFirstBook(Library & l){
22     return (l.numBooks > 0) ? l.titles[0] : "None";
23 }
24
25
```

pointer

Memory Management

Local memory on the stack is managed by the computer

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

Understand when and how to use reference (&) and dereference (*) operators

Tip: If you can, avoid using **new** at all!

Memory Management

You are building a search tool over a collection of very large image files. One operation you want is to search an image for a particular pixel pattern (and return whether it exists or not). Assuming the query pattern and the input image are both of type **Image**, what might our function header look like?

bool findImage(const Image& query, const Image& Galaxy)

const Image & Galaxy

The Const Keyword

Const means that an object cannot be modified

Variables `const int x = 5;`

Pointers `const int *` → can't change val / `int * const` → can't change pointer address

Reference `const int &`

Method `Class::GetImage() const`

Pointer-to-constant vs constant pointer

Skipped Slide!

```
1  int x = 3;
2  int y = 2;
3  // *** A ***
4  const int* a = &x;
5
6
7  a = &y;
8
9  // *** B ***
10 const int* b = &x;
11
12
13 *b = y;
14
15 // *** C ***
16 int* const c = &x;
17
18
19 c = &y;
20
21 // *** D ***
22 int* const d = &x;
23
24
25 *d = y;
```

These are ok! why?

These are invalid! Why?

Const pointers vs const methods

```
1  struct BlackBox {
2      void update(const int & obj) {
3          myVal = obj;
4
5          obj++;
6
7      }
8
9      void update(int & obj) const {
10         myVal = obj;
11
12         obj++;
13
14     }
15
16     void update(const int & obj) const {
17         myVal = obj;
18
19         obj++;
20
21     }
22
23     int myVal;
24
25 };
```

Templates

template1.cpp

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

List Abstract Data Type

A list is an **ordered** collection of items

- Items can be either **heterogeneous** or **homogenous**
- The list can be of a **fixed size** or is **resizable**

What types of “stuff” do we want in our list?