Data Structures

C++ Review

CS 225

August 23, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

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

no rec letter needed !

student run

apply for fall

PROMOTING UNDERGRADUATE RESEARCH IN ENGINEERING

Scan for:

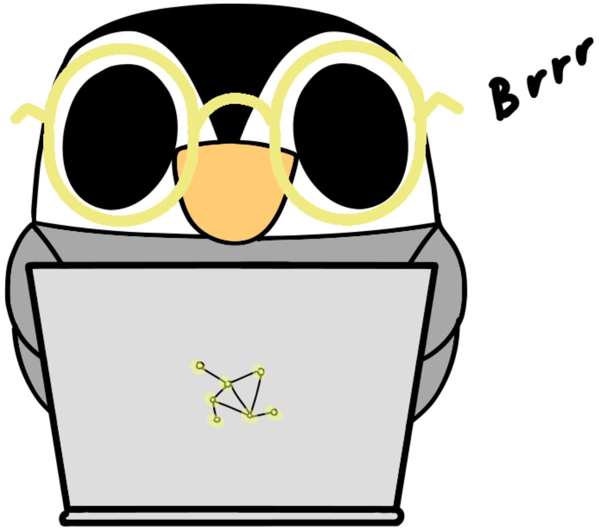
- Interest form ✉
- Website 🌐
- Discord

(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

Drafting a ‘Library’ class

```
class Library {
public:













private:










```

Class Fundamentals

Constructor

Destructor

Class Fundamentals

Does our library class need a destructor?

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.

2.

3.

```
class Library {
public:
    int numBooks;
    std::string * titles;
    ~Library();
    Library( int num, std::string* list );
};

Library::~Library(){
   delete titles;
   titles = nullptr;
}

Library::Library(int num, std::string* list){
    numBooks = inNum;
    titles = new std::string[ inNum ];
    std::copy(inList, inList + inNum, titles);
}

int main(){
   std::string myBooks[3] = {"A", "B", "C"};
   Library L1( 3, myBooks );
   Library L2( L1 );
   return 0;
}
```

```cpp
class Library {
public:
    int numBooks;
    std::string * titles;
    ~Library();
    Library( int num, std::string* list );
};

Library::~Library(){
   delete titles;
   titles = nullptr;
}

Library::Library(int num, std::string* list){
    numBooks = inNum;
    titles = new std::string[ inNum ];
    std::copy(inList, inList + inNum, titles);
}

int main(){
   std::string myBooks[3] = {"A", "B", "C"};
   Library L1( 3, myBooks );
   Library L2( L1 );
   return 0;
}
```

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

Heap

Global

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 (*)

Memory Management - Parameters

Value

Value — Pointer

Reference

Memory Management - Parameters

```
class Library {
public:
    int numBooks;
    std::string * titles;
};

// *** Function A ***
std::string getFirstBook(Library l){
    return (l.numBooks > 0) ? l.titles[0] : "None";
}

// *** Function B ***
std::string getFirstBook(Library * l){
    return(l->numBooks > 0) ? l->titles[0] : "None";
}

// *** Function C ***
std::string getFirstBook(Library & l){
    return (l.numBooks > 0) ? l.titles[0] : "None";
}

```

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?

The Const Keyword

Const means that an object cannot be modified

Variables

Pointers

Reference

Method

Pointer-to-constant vs constant pointer

```
int x = 3;
int y = 2;
// *** A ***
const int* a = &x;

a = &y;

// *** B ***
const int* b = &x;

*b = y;

// *** C ***
int* const c = &x;

c = &y;

// *** D ***
int* const d = &x;

*d = y;
```

Const pointers vs const methods

```
struct BlackBox {
    void update(const int & obj) {
        myVal = obj;

        obj++;
    }

    void update(int & obj) const {
        myVal = obj;

        obj++;
    }

    void update(const int & obj) const {
        myVal = obj;

        obj++;
    }

    int myVal;
};
```

Templates

template1.cpp

```cpp


T maximum(T a, T b) {
  T result;
  result = (a > b) ? a : b;
  return result;
}
```

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?

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