

# CSE 333 – SECTION 5

---

C++

# Section Feedback

- Learn to throw
- More:
  - Candy/food
  - Jokes
  - Serious
  - Spanish
- Tools/Tricks/Tips
- Whiteboard Diagrams are...
  - Helpful
  - Not helpful
    - Yet this person copies one on the back of the index card...

# Overview

- C++ Classes, Constructors, new, delete, etc.
- Drawing Memory Diagrams

# C++ classes

- Encapsulation and Abstraction
- Access specifiers:
  - Public: anything outside the class can access it
  - Protected: only this class and derived classes can access it
  - Private: only this class can access it
- Polymorphism
- Multiple Inheritance

# new and delete

- new is used to allocate objects and primitive data types on the heap
- delete is used to deallocate these heap allocated objects
- Use “delete [ ] array” on an array
- Unlike malloc() and free(), new and delete are operators

# Initialization vs Assignment

```
#define MAXSIZE 3

class IntArrayList {
public:
    IntArrayList() : array_(new int[MAXSIZE]), len_(0), maxsize_(MAXSIZE) { }

    IntArrayList(const int *const arr, size_t len) : len_(len), maxsize_(len*2) {
        array_ = new int[maxsize_];
        memcpy(array_, arr, len * sizeof(int));
    }

    IntArrayList(const IntArrayList &rhs) {
        len_ = rhs.len_;
        maxsize_ = rhs.maxsize_;
        array_ = new int[maxsize_];
        memcpy(array_, rhs.array_, maxsize_ * sizeof(int));
    }
    ...
private:
    int *array_;
    size_t len_;
    size_t maxsize_;
};
```

# Memory diagram

- See: `wrapmain.cc` && `IntArrayList.h`
- What does memory look like when you call the default constructor?
- How about the copy constructor?

# Memory diagram

```
class Wrap {  
public:  
    Wrap() : p_(nullptr) { }  
    Wrap(IntArrayList *p)  
        : p_(p) { *p_ = *p; }  
    IntArrayList *p() const  
        { return p_; }  
private:  
    IntArrayList *p_;  
}  
  
struct List {  
    IntArrayList v;  
}
```

```
int main() {  
    Wrap a;  
    Wrap b(new IntArrayList);  
    struct List c { };  
    struct List d {*b.p()};  
    a = b;  
    c = d;  
    Wrap *e;  
    e = &a;  
    Wrap *f = new Wrap(&d.v);  
    struct List *g =  
        new struct List;  
    g->v = *(new IntArrayList);  
    delete f;  
    delete g;  
    return 0;  
}
```



# Operator Overloading

- A form of polymorphism.
- Give special meanings to operators in user-defined classes
- Special member functions in classes with a particular naming convention
- For E.g., for overloading the '+' operator, define a member function named operator+

# Common operators

- The most commonly overloaded operators are
  - = (assignment operator)
  - + - \* (binary arithmetic operators)
  - += -= \*= (compound assignment operators)
  - == != (comparison operators)

# Exercise 1

- A) Create a Memory Diagram for the following code:

```
int main() {  
    IntArrayList a;  
    IntArrayList *b = new IntArrayList();  
    struct List l { a };  
    struct List m { *b };  
    Wrap w(b);  
    delete b;  
}
```

- B) Identify any potential leaks (if any)

# Exercise 2

- Modify `wrapmain.cc` and `IntArrayList.h` such that there are no memory leaks in `wrapmain.cc`