CSCI 200: Foundational Programming Concepts & Design Lecture 22

Memory Management via The Big Three

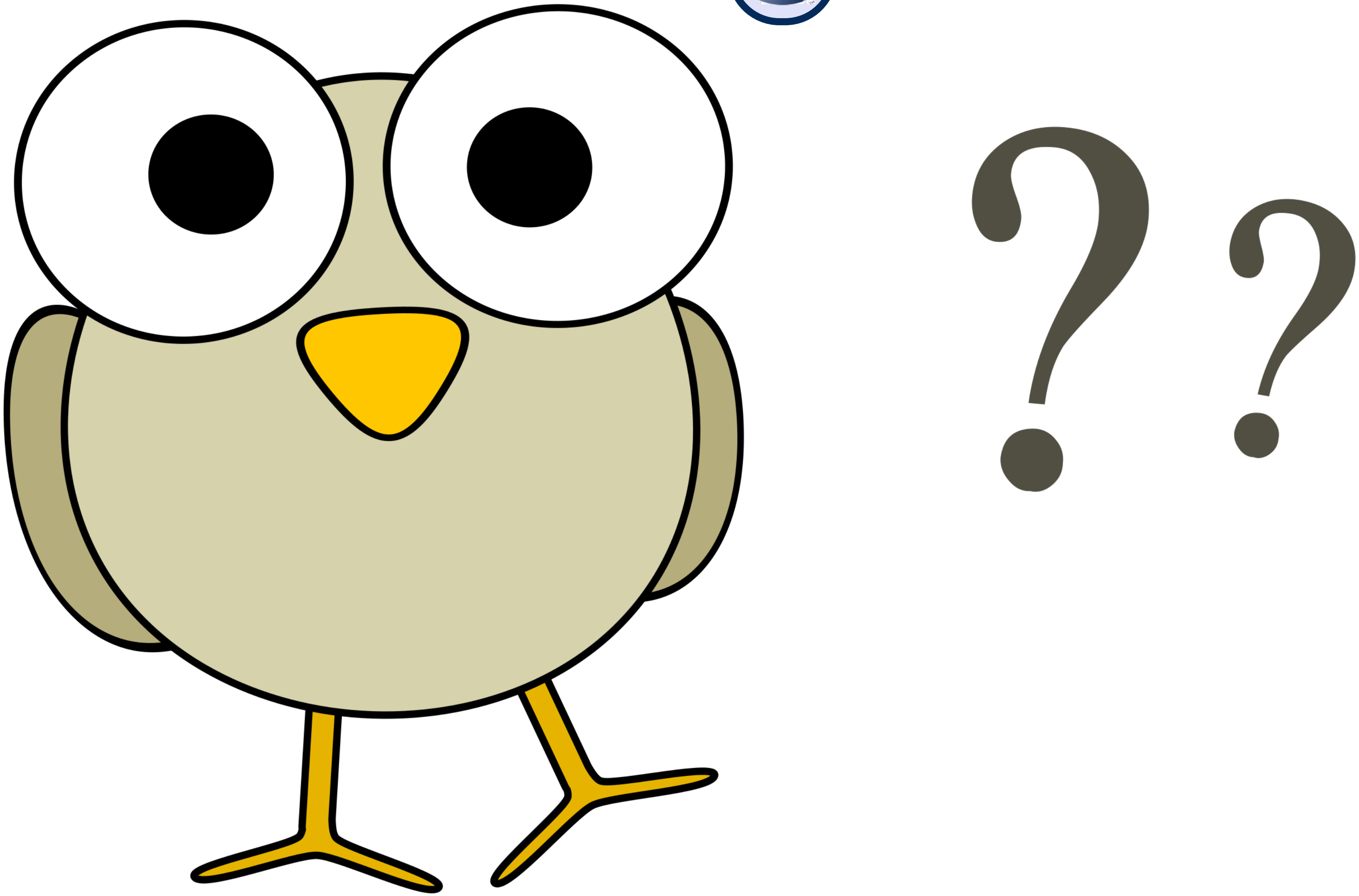
Deep v Shallow Copy

Previously in CSCI 200

- Pass-by-Value
  vs Pass-by-Pointer
  vs Pass-by-Reference

- **`new`** and **`delete`**

Questions?



Example Box Class

```
// Box.h
class Box {
public:
    Box(const int SIZE);
    int getBoxSize() const;
private:
    int _size;
};
```

```
// Box.cpp
#include "Box.h"

Box::Box(const int SIZE) {
  _size = SIZE;
}
int Box::getBoxSize() const {
  return _size;
}
```

Example Warehouse Class

```cpp
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};
```

```cpp
// Warehouse.cpp
#include "Warehouse.h”

Warehouse::Warehouse() {
    _pBoxen = new vector<Box*>;
}

void Warehouse::storeInBox(const int SIZE) {
    _pBoxen->push_back(new Box(SIZE+1));
}

Box* Warehouse::getBox(const int POS) {
  return _pBoxen->at(POS);
}

int Warehouse::getNumberBoxes() const {
  return _pBoxen->size();
}
```

```cpp
// main.cpp

Warehouse *pWarehouseH = new Warehouse;    // new calls constructor

pWarehouseH->storeInBox(4);
```

Learning Outcomes For Today

- Define, list, and implement the Big 3.
- Explain the difference between a shallow copy and a deep copy. Implement both.
- Overload common operators and discuss reasons why operator overloading is useful.

On Tap For Today

- Operator Overloading
- Assignment
  - Copy
    - Shallow vs. Deep
- The Big 3
- Practice

On Tap For Today

- Operator Overloading
- Assignment
    - Copy
        - Shallow vs. Deep
- The Big 3
- Practice

Example Warehouse Class

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};
```

```
// Warehouse.cpp
#include "Warehouse.h”
Warehouse::Warehouse() {
    _pBoxen = new vector<Box*>;
}
void Warehouse::storeInBox(const int SIZE) {
    _pBoxen->push_back(new Box(SIZE+1));
}
Box* Warehouse::getBox(const int POS) {
  return _pBoxen->at(POS);
}
int Warehouse::getNumberBoxes() const {
  return _pBoxen->size();
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse;
pWarehouseH->storeInBox(4);
cout << pWarehouseH << endl;     // what happens?
```

Example Warehouse Class

```cpp
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};
```

```cpp
// Warehouse.cpp
#include "Warehouse.h”
Warehouse::Warehouse() {
    _pBoxen = new vector<Box*>;
}
void Warehouse::storeInBox(const int SIZE) {
    _pBoxen->push_back(new Box(SIZE+1));
}
Box* Warehouse::getBox(const int POS) {
  return _pBoxen->at(POS);
}
int Warehouse::getNumberBoxes() const {
  return _pBoxen->size();
}
```

```cpp
// main.cpp
Warehouse *pWarehouseH = new Warehouse;
pWarehouseH->storeInBox(4);
cout << pWarehouseH << endl;     // prints address 0x42dc28ad
```

Example Warehouse Class

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};
```

```
// Warehouse.cpp
#include "Warehouse.h”
Warehouse::Warehouse() {
    _pBoxen = new vector<Box*>;
}
void Warehouse::storeInBox(const int SIZE) {
    _pBoxen->push_back(new Box(SIZE+1));
}
Box* Warehouse::getBox(const int POS) {
  return _pBoxen->at(POS);
}
int Warehouse::getNumberBoxes() const {
  return _pBoxen->size();
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse;
pWarehouseH->storeInBox(4);
cout << pWarehouseH << endl;     // prints address 0x42dc28ad
cout << *pWarehouseH << endl;    // what happens?
```

Example Warehouse Class

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};
```

```
// Warehouse.cpp
#include "Warehouse.h”
Warehouse::Warehouse() {
    _pBoxen = new vector<Box*>;
}
void Warehouse::storeInBox(const int SIZE) {
    _pBoxen->push_back(new Box(SIZE+1));
}
Box* Warehouse::getBox(const int POS) {
  return _pBoxen->at(POS);
}
int Warehouse::getNumberBoxes() const {
  return _pBoxen->size();
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse;
pWarehouseH->storeInBox(4);
cout << pWarehouseH << endl;     // prints address 0x42dc28ad
cout << *pWarehouseH << endl;    // ERROR!
// invalid operands to binary expression ('std::__1::ostream’ (aka
// 'basic_ostream<char>') and 'Warehouse')
```

Operator Overloading

- What does overloading mean?
- What operators do we have?

Precedence	Operator	Associativity
1	Parenthesis: ( )	Innermost First
2	Scope Resolution: S::	Left to Right
3	Postfix Unary Operators: a++ a-- a[ ] a. f() p->	
4	Prefix Unary Operators: ++a --a +a -a !a (type)a &a *p new delete	Right to Left
5	Binary Operators: a*b a/b a%b	Left to Right
6	Binary Operators: a+b a-b	
7	Relational Operators: a<b a>b a<=b a>=b	
8	Relational Operators: a==b a!=b	
9	Logical Operators: a&&b	
10	Logical Operators: a||b	
11	Assignment Operators: a=b a+=b a-=b a*=b a/=b a%=b	Right to Left

Operator Overloading

- What does overloading mean?
- What operators do we have?
- Which operators can we overload?

Precedence	Operator	Associativity
1	Parenthesis: ( )	Innermost First
2	Scope Resolution: S::	Left to Right
3	Postfix Unary Operators: a++ a-- a[ ] a. f() p->	
4	Prefix Unary Operators: ++a --a +a -a !a (type)a &a *p new delete	Right to Left
5	Binary Operators: a*b a/b a%b	Left to Right
6	Binary Operators: a+b a-b	
7	Relational Operators: a<b a>b a<=b a>=b	
8	Relational Operators: a==b a!=b	
9	Logical Operators: a&&b	
10	Logical Operators: a||b	
11	Assignment Operators: a=b a+=b a-=b a*=b a/=b a%=b	Right to Left

Operator Overloading

- What does overloading mean?
- What operators do we have?
- Which operators can we overload?
- And more
  - <<
  - >>
  - (and others too)

Printing the Warehouse

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};

std::ostream& operator<<(
  std::ostream&, const Warehouse&
);
```

```
// Warehouse.cpp
#include "Warehouse.h”

/* ... */

std::ostream& operator<<(
  std::ostream& os, const Warehouse& WH
) {
  os << "Warehouse has "
     << WH.getNumberBoxes() << " boxes";
  return os;
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse;
pWarehouseH->storeInBox(4);
cout << pWarehouseH << endl;     // prints address 0x42dc28ad
cout << *pWarehouseH << endl;    // prints "Warehouse has 1 boxes"
```

Now What?

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};

std::ostream& operator<<(
  std::ostream&, const Warehouse&
);
```

```
// Warehouse.cpp
#include "Warehouse.h”

/* ... */

std::ostream& operator<<(
  std::ostream& os, const Warehouse& WH
) {
  os << "Warehouse has "
     << WH.getNumberBoxes() << " boxes";
  return os;
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse, *pWarehouseC = new Warehouse;
pWarehouseH->storeInBox(4);
pWarehouseC->storeInBox(2);
pWarehouseC = pWarehouseH;        // what does this do?
```

Now What?

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};

std::ostream& operator<<(
  std::ostream&, const Warehouse&
);
```

```
// Warehouse.cpp
#include "Warehouse.h”

/* ... */

std::ostream& operator<<(
  std::ostream& os, const Warehouse& WH
) {
  os << "Warehouse has "
     << WH.getNumberBoxes() << " boxes";
  return os;
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse, *pWarehouseC = new Warehouse;
pWarehouseH->storeInBox(4);
pWarehouseC->storeInBox(2);
*pWarehouseC = *pWarehouseH;     // what does this do?
```

On Tap For Today

- Operator Overloading
- Assignment
  - Copy
    - Shallow vs. Deep
- The Big 3
- Practice

Assignment

- Generally

```
lhs = rhs
```

- Assign the right hand side to the left hand side

On Tap For Today

- Operator Overloading
- Assignment
  - Copy
    - Shallow vs. Deep
- The Big 3
- Practice

Copying

- Performed with two `lvalue`s that are both backed by memory

- Can be done in two ways
  1. Reuse existing memory
  2. Duplicate memory

- AKA Shallow Copy or Deep Copy

On Tap For Today

- Operator Overloading
- Assignment
  - Copy
    - Shallow vs. Deep
- The Big 3
- Practice

Shallow Copy vs. Deep Copy

- Shallow Copy: create new `lvalue` backed by same memory
- Deep Copy: create new `lvalue` with new memory

Shallow Copy vs. Deep Copy

- Shallow Copy: create new `lvalue` backed by same memory
  - Makes a new alias

- Deep Copy: create new `lvalue` with new memory
  - Makes a new instance

Shallow Copy? Deep Copy?

```
// Warehouse.h
class Warehouse {
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
private:
    std::vector<Box*>* _pBoxen;
};

std::ostream& operator<<(
  std::ostream&, const Warehouse&
);
```

```
// Warehouse.cpp
#include "Warehouse.h”

/* ... */

std::ostream& operator<<(
  std::ostream& os, const Warehouse& WH
) {
  os << "Warehouse has "
     << WH.getNumberBoxes() << " boxes";
  return os;
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse, *pWarehouseC = new Warehouse;
pWarehouseH->storeInBox(4);
pWarehouseC->storeInBox(2);
pWarehouseC = pWarehouseH;       // shallow or deep?
*pWarehouseC = *pWarehouseH;     // shallow or deep?
```

Specify Copy Assignment Operator

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
    Warehouse& operator=(
      const Warehouse&
    );
private:
    std::vector<Box*>* _pBoxen;
};
std::ostream& operator<<( ... );
```

```
// Warehouse.cpp
#include "Warehouse.h”

/* ... */

Warehouse& Warehouse::operator=(
  const Warehouse& OTHER
) {
  // guard against self assignment
  if(this == &OTHER) return *this;

  // delete existing contents

  // perform deep copy from OTHER to this

  return *this;
}
```

```
// main.cpp
Warehouse *pWarehouseH = new Warehouse, *pWarehouseC = new Warehouse;
pWarehouseH->storeInBox(4);
pWarehouseC->storeInBox(2);
pWarehouseC = pWarehouseH;      // shallow by definition
*pWarehouseC = *pWarehouseH;    // deep by overloaded definition
```

Now what happens?

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
    Warehouse& operator=(
      const Warehouse&
    );
private:
    std::vector<Box*>* _pBoxen;
};
std::ostream& operator<<( ... );
```

```
// Warehouse.cpp
#include "Warehouse.h"

// constructor

// copy assignment operator

// methods to use class
```

```
void someFunction(Warehouse wh) { /* ... */ }

someFunction( Warehouse() );      // what gets called?
```

The What?

```
void someFunction(Warehouse wh) { /* ... */ }

someFunction( Warehouse() );       // what gets called?


// main.cpp
someFunction( Warehouse() );        // the constructor to make Warehouse
                                    // the copy constructor to make wh
```

Copy Constructor

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    Warehouse(const Warehouse&);
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
    Warehouse& operator=(
      const Warehouse&
    );
private:
    std::vector<Box*>* _pBoxen;
};
std::ostream& operator<<( ... );
```

```
// Warehouse.cpp
#include "Warehouse.h”

// constructor

// copy constructor
Warehouse::Warehouse(const Warehouse& OTHER) {
  // perform deep copy from OTHER to this
}

// copy assignment operator

// methods to use class
```

```
Warehouse warehouseH;
warehouseH.storeInBox(4);

Warehouse warehouseC( warehouseH ); // copy constructor

Warehouse warehouseD;                // initialize w/ default constructor
warehouseD = warehouseH;             // copy assignment operator
```

Copy Constructor

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    Warehouse(const Warehouse&);
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
    Warehouse& operator=(
      const Warehouse&
    );
private:
    std::vector<Box*>* _pBoxen;
};
std::ostream& operator<<( ... );
```

```
// Warehouse.cpp
#include "Warehouse.h”

// constructor

// copy constructor
Warehouse::Warehouse(const Warehouse& OTHER) {
  // perform deep copy from OTHER to this
}

// copy assignment operator

// methods to use class
```

```
Warehouse *pWarehouseH = new Warehouse;
pWarehouseH->storeInBox(4);

Warehouse *pWarehouseC = new Warehouse( *pWarehouseH ); // copy constructor

Warehouse *pWarehouseD = new Warehouse(); // initialize w/ default constructor
*pWarehouseD = *pWarehouseH;                 // copy assignment operator
```

Cleanup Time

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    Warehouse(const Warehouse&);
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
    Warehouse& operator=(
      const Warehouse&
    );
private:
    std::vector<Box*>* _pBoxen;
};
std::ostream& operator<<( ... );
```

```
// Warehouse.cpp
#include "Warehouse.h”

// constructor

// copy constructor
Warehouse::Warehouse(const Warehouse& OTHER) {
  // perform deep copy from OTHER to this
}

// copy assignment operator

// methods to use class
```

```
Warehouse *pWarehouseH = new Warehouse; // new + constructor allocates memory

pWarehouseH->storeInBox(4);             // storing in a box allocates memory

delete pWarehouse;                      // dellocate all that memory
```

Removing object calls Destructor

```
// Warehouse.h
class Warehouse {
public:
    Warehouse();
    Warehouse(const Warehouse&);
    ~Warehouse();
    void storeInBox(int)
    Box* getBox(int);
    int getNumberBoxes() const;
    Warehouse& operator=(
      const Warehouse&
    );
private:
    std::vector<Box*>* _pBoxen;
};
std::ostream& operator<<( ... );
```

```
// Warehouse.cpp
#include "Warehouse.h”

// constructor

// copy constructor

// destructor
Warehouse::~Warehouse() {
  // delete entire contents of object
}

// copy assignment operator

// methods to use class
```

```
Warehouse *pWarehouseH = new Warehouse; // new + constructor allocates memory

pWarehouseH->storeInBox(4);             // storing in a box allocates memory

delete pWarehouse;                      // dellocate all that memory
```

On Tap For Today

- Operator Overloading
- Assignment
  - Copy
    - Shallow vs. Deep
- The Big 3
- Practice

The Big 3

- The Big 3
  - Destructor (default: delete references)
  - Copy Assignment Operator (default: shallow)
  - Copy Constructor (default: shallow)

- Rule of 3
  - If you explicitly make one of them, you should explicitly make all three

Object Lifecycle

- Where do the following fit into an object’s life cycle? When are each applied?
  - Constructor
  - Copy Assignment
  - Destructor

Getter Beware!

- Consider this scenario

What gets printed?

```
class InnerClass {
public:
   InnerClass() { x = 1; }
   int x;
};
class OuterClass {
public:
   InnerClass getIC();
private:
   InnerClass mIc;
};
```

```
int main() {
  OuterClass oc;
  cout << oc.getIC().x << endl;
  oc.getIC().x = 5;
  cout << oc.getIC().x << endl;
  return 0;
}
```

```
1
1
```

What gets printed?

```
class InnerClass {
public:
  InnerClass() { x = 1; }
  int x;
};
class OuterClass {
public:
  InnerClass getIC();
private:
  InnerClass mIc;
};
```

```
int main() {
  OuterClass oc;
  cout << oc.getIC().x << endl;
  InnerClass ic = oc.getIC();
  ic.x = 5;
  cout << oc.getIC().x << endl;
  return 0;
}
```

```
1
1
```

What gets printed?

```
class InnerClass {
public:
   InnerClass() { x = 1; }
   int x;
};
class OuterClass {
public:
   InnerClass* getIC();
private:
   InnerClass* mpIc;
};
```

```
int main() {
  OuterClass oc;
  cout << oc.getIC()->x << endl;
  InnerClass* ic = oc.getIC();
  ic->x = 5;
  cout << oc.getIC()->x << endl;
  return 0;
}
```

```
1
5
```

What gets printed?

```
class InnerClass {
public:
   InnerClass() { x = 1; }
   int x;
};
class OuterClass {
public:
   InnerClass* getIC();
private:
   InnerClass* mpIc;
};
```

```
int main() {
   OuterClass oc;
   cout << oc.getIC()->x << endl;
   oc.getIC()->x = 5;
   cout << oc.getIC()->x << endl;
   return 0;
}
```

```
1
5
```

On Tap For Today

- Operator Overloading
- Assignment
  - Copy
    - Shallow vs. Deep
- The Big 3
- Practice

To Do For Next Time

- Can begin L4A
- No class Monday
- Quiz 4 on Wednesday – OOP + Pointers