

# CSC 211: Object Oriented Programming

## Copy Constructors and Assignment Operator

Marco Alvarez

Department of Computer Science and Statistics  
University of Rhode Island

Fall 2019



## More on constructors ...

- So far ...
  - ✓ default constructors, overloaded constructors
- C++ also defines **copy constructors**
  - ✓ used to create an object as a copy of an existing object
  - ✓ if you don't define your own, C++ will synthesize one copy constructor for you

```
Point2D obj1;           // default constructor
Point2D obj2(4.5, 3.2); // overloaded constructor
Point2D obj3(obj2);     // copy constructor
Point2D obj4 = obj3;    // copy constructor
```

2

## When are copy constructors invoked?

```
Point2D myfunc(Point2D obj) {
    Point2D newobj;
    // ...
    return newobj;
}

int main () {
    // copy constructor is invoked when an object is initialized from
    // another object of the same type
    Point2D obj2(4.5, 3.2);    // overloaded constructor
    Point2D obj3(obj2);        // copy constructor
    Point2D obj4 = obj3;       // copy constructor

    // copy constructor is invoked when a non-reference object is
    // passed to a function
    myfunc(obj4);              // copy constructor

    // copy constructor is invoked when a non-reference object is
    // returned from a function
    Point2D obj5 = myfunc(obj2);
}
```

3

## Shallow vs deep copies

- Synthesized copy constructors perform **shallow copies**
    - ✓ a shallow copy is a byte-to-byte copy of all data members (works fine most of the cases, except when pointers are used)
- ```
Point2D::Point2D(const Point2D& obj) {
    x = obj.x;
    y = obj.y;
    // ...
}
```
- Sometimes a **deep copy** is necessary (can handle more complex objects)
    - ✓ must define your own copy constructor

4

```

class Array {
public:
    Array(int cap);
    ~Array();
private:
    int size;
    int capacity;
    int *ptr;
};

```

```

Array::Array(int cap) {
    size = 0;
    capacity = cap;
    ptr = new int[cap];
}

```

```

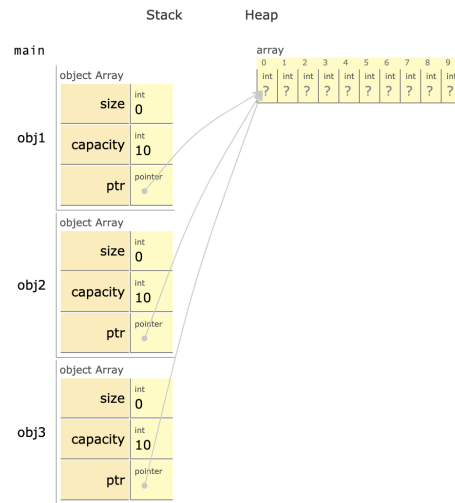
Array::~Array() {
    delete [] ptr;
}

```

```

int main () {
    Array obj1(10);
    Array obj2(obj1);
    Array obj3 = obj2;
}

```



shallow copies

5

```

Array::Array(int cap) {
    size = 0;
    capacity = cap;
    ptr = new int[cap];
}

```

```

Array::Array(Array& obj) {
    size = obj.size;
    capacity = obj.capacity;
    ptr = new int[capacity];
    for (int i = 0 ; i < size ; i++) {
        ptr[i] = obj.ptr[i];
    }
}

```

```

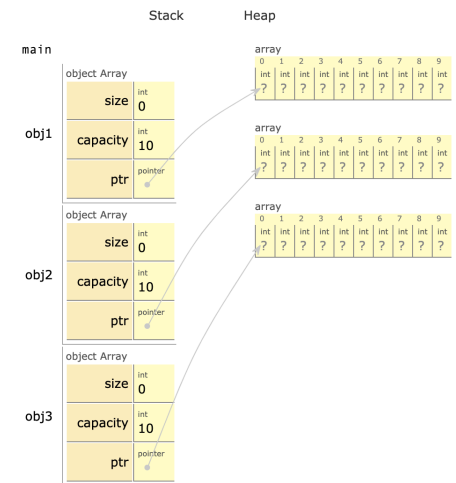
Array::~Array() {
    delete [] ptr;
}

```

```

int main () {
    Array obj1(10);
    Array obj2(obj1);
    Array obj3 = obj2;
}

```



deep copies

6

## The assignment operator =

- Assignment is not construction
- The assignment operator '=' assigns an object to an existing object (already constructed)

```

Point2D obj1;           // default constructor
Point2D obj2(4.5, 3.2); // overloaded constructor
Point2D obj3(obj2);     // copy constructor
Point2D obj4 = obj3;    // copy constructor
obj1 = obj4;            // assignment operator

```

- If you don't define your own, C++ will synthesize one assignment operator for you (performs **shallow copy**)

7

## How to overload the '=' operator?

```

Point2D& Point2D::operator=(const Point2D &obj) {
    // always check against self-assignment
    // especially when performing deep copies
    if (this != &obj) {
        x = obj.x;
        y = obj.y;
    }
    // always return *this, necessary for
    // cascade assignments (a = b = c)
    return *this;
}

```

can perform either shallow or deep copies

8

## The **this** pointer

- Pointer accessible only within member functions of a class
  - ✓ it points to the object for which the member function is called
  - ✓ static member functions do not have this pointer

```
void Date::set_year(int y) {  
    // statements below are equivalent  
    year = y;  
    this->year = y;  
    (*this).year = y;  
}
```

9

## How many calls?

```
Point2D myfunc(const Point2D& obj) {  
    Point2D newobj;  
    newobj = obj;  
    // ...  
    return newobj;  
}  
  
int main () {  
    Point2D obj2(4.3, 1.1);  
    Point2D obj3(obj2);  
    Point2D obj4 = myfunc(obj3);  
    Point2D obj5;  
    obj5 = obj4 = obj2;  
}
```

10

Live Example  
(matrix)