

CSC 211: Computer Programming

Copy Constructors and Operator Overloading

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Summer 2023



Original design and development by Dr. Marco Alvarez

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 (to initialize parameter)
    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;
}

```

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;
}

```

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

## 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;
}

```

8

# Overloading Operators

## How to overload the '=' operator?

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

Modify the self object reference and return it

can perform either shallow or deep copies

10

## How many copy constructor 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;
}
```

11

## Static Data Members

- We can define static class members using **static** keyword
- When we declare a member of a class as static it means **no matter how many objects of the class are created, there is only one copy of the static member**
- A static member is **shared by all objects** of the class

12

# Static Data Members

- All static data is initialized to zero when the first object is created, if no other initialization is present
- We can't initialize static members in the class definition
- initialized outside the class

[https://www.tutorialspoint.com/cplusplus/cpp\\_static\\_members.htm](https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm)Source:

13

# Static Data Members

```
#include <iostream>
using namespace std;

class Box {
public:
 static int objectCount;

 // Constructor definition
 Box(double l = 2.0, double w = 2.0, double h = 2.0) {
 cout << "Constructor called." << endl;
 length = l;
 width = w;
 height = h;

 // Increase every time object is created
 objectCount++;
 }
 double Volume() {
 return length * width * height;
 }

private:
 double length; // Length of a box
 double width; // width of a box
 double height; // Height of a box
};
```

14

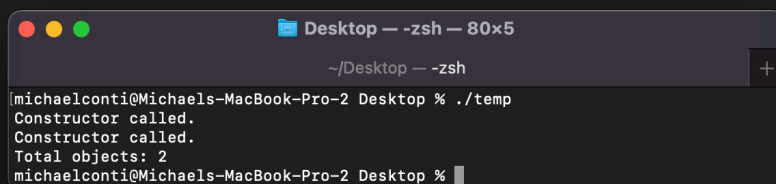
# Static Data Members

```
// Initialize static member of class Box
int Box::objectCount = 0;

int main() {
 Box Box1(3.3, 1.2, 1.5); // Construct box1
 Box Box2(8.5, 6.0, 2.0); // Construct box2

 // Print total number of objects.
 cout << "Total objects: " << Box::objectCount << endl;

 return 0;
}
```



```
Desktop -- -zsh -- 80x5
~/Desktop -- -zsh
michaelconti@Michaels-MacBook-Pro-2 Desktop % ./temp
Constructor called.
Constructor called.
Total objects: 2
michaelconti@Michaels-MacBook-Pro-2 Desktop %
```

15

# Static Member Functions

- By declaring a function member as static, you make it independent of any particular object of the class
- A static member function can be called even if no objects of the class exist
- Static functions are accessed using only the class name and the scope resolution operator ::

[https://www.tutorialspoint.com/cplusplus/cpp\\_static\\_members.htm](https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm)Source:

16

## Static Member Functions

- A static member function can only access:
  - static data member
  - other static member functions
  - any other functions from outside the class
- You could use a static member function to determine whether some objects of the class have been created or not

[https://www.tutorialspoint.com/cplusplus/cpp\\_static\\_members.htm](https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm)Source:

17

## Static Member Functions

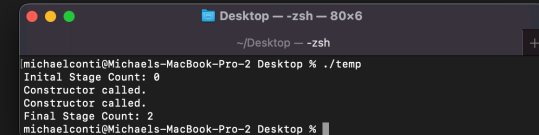
```
static int Box::getCount() {
 return objectCount;
}

int main() {
 // Print total number of objects before creating object.
 cout << "Initial Stage Count: " << Box::getCount() << endl;

 Box Box1(3.3, 1.2, 1.5); // Construct box1
 Box Box2(8.5, 6.0, 2.0); // Construct box2

 // Print total number of objects after creating object.
 cout << "Final Stage Count: " << Box::getCount() << endl;

 return 0;
}
```



```
Desktop - zsh - 80x6
~/Desktop - zsh
michaelconti@Michaels-MacBook-Pro-2 Desktop % ./temp
Initial Stage Count: 0
Constructor called.
Constructor called.
Final Stage Count: 2
michaelconti@Michaels-MacBook-Pro-2 Desktop %
```

18

## Lets try it

- Overload the ++ (post increment) operator for the Date class we build in a previous lecture to increment the year for any object of type Date
  - Hint: 

```
void operator++(int);
Date someDay(12, 30, 1993);
someDay++;
```
- Write a (shallow) copy constructor for objects of type Date
- Include a static member that counts the number of date objects

19