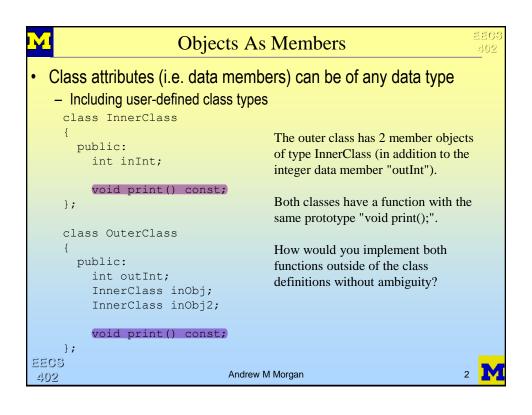


EECS402 Lecture 07

Andrew M. Morgan

Savitch Ch. 7
Objects
Constructors
Destructors

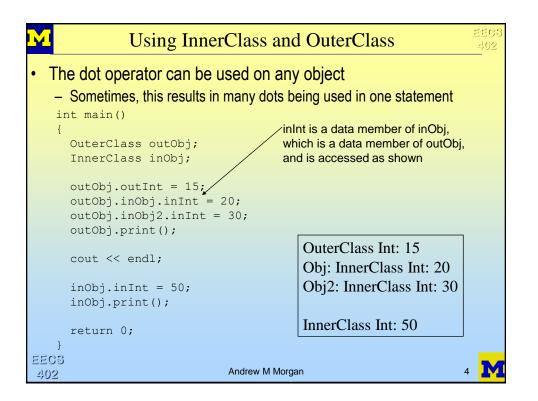


```
Example Implementation
 While the function prototypes are the same, they are unique in
  what class they belong to

    The use of scope resolution can solve the ambiguity problem

    Remember – scope resolution is often read "belongs to"

                                         This version of print() "belongs to"
                                          the class InnerClass
       void InnerClass::print() const
         cout << "InnerClass Int: " << inInt << endl;</pre>
                                           This version of print() "belongs to"
                                            the class OuterClass
       void OuterClass::print() const
         cout << "OuterClass Int: " << outInt << endl;</pre>
         cout << "Obj: ";
         inObj.print();
         cout << "Obj2: ";
         inObj2.print();
EECS
                                Andrew M Morgan
703
```





Constructors & Destructors

- A constructor is a special member function of a class
 - The name of the function is the same as the name of the class
 - It has no return type (can not return a value)
 - It is automatically called every time a new object is constructed
 - The programmer can not directly "call" the constructor
 - Often abbreviated "ctor"
- A destructor is a special member function of a class
 - The name of the function is the same as the name of the class with a "~" before it
 - It has <u>no</u> return type (can not return a value)
 - It is *automatically* called when an object is destroyed
 - · When a static object goes out of scope, or a dynamic object is deleted (will discuss
 - The programmer can not directly "call" the destructor
 - Often abbreviated "dtor"

EECS 402

Andrew M Morgan





Ctor And Dtor Purpose

- Why are ctors important?
 - Since they are called automatically every time an object is created, they are usually used to initialize data members for a new object
 - Ctors, like most functions can be overloaded
 - · Developer might provide several different ctors (with unique signatures) to initialize members in different ways
 - The "default ctor" is the ctor that takes in no parameters
- Why are dtors important?
 - At the moment, they aren't
 - They are very important when using dynamic allocation, and more detail will be given at that time
 - Dtors, unlike most functions, can not be overloaded
 - Only one dtor can exist per class!

EECS 402

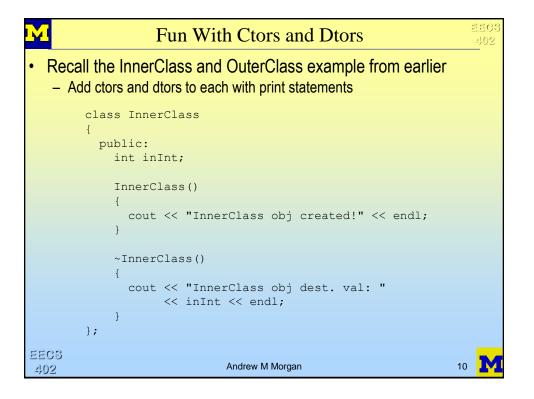
Andrew M Morgan



```
Ctors and Dtors - Prototypes
 Programmers do not call constructors or destructors
 They are called automatically!
class IntClass
 private:
   int val;
 public:
   IntClass(); //Default constructor
   IntClass(const int inVal); //Ctor to assign val to
                              //a specific value
   ~IntClass(); //Destructor
   void add(const int addVal); //Adds "addVal" to the data member
   void sub(const int subVal); //Subtracts "subVal" from data member
};
EECS
                              Andrew M Morgan
402
```

```
Ctors and Dtors, Implementation
     //Default constructor - NO return type (not even void)
     //Called when space for an IntClass object is claimed
     IntClass::IntClass()
       cout << "IntClass obj constructed!" << endl;</pre>
     //Ctor to assign data member to specified value
    IntClass::IntClass(const int inVal)
      val = inVal;
      cout << "IntClass obj constructed!" << endl;</pre>
     //Destructor - called when an IntClass object is destroyed
     IntClass::~IntClass()
      cout << "IntClass obj destroyed - val was: "</pre>
            << val << endl;
EECS 1
                            Andrew M Morgan
402
```

```
Use of Ctors and Dtor
         int main (void)
           IntClass i1;
                              //Initializes il.val to 0
            IntClass i2(5); //Initializes i2.val to 5
            cout << "Done with declarations." << endl;</pre>
            i1.add(50); //Adds 50 to i1.val, resulting in 50
           i2.add(75); //Adds 75 to i2.val, resulting in 80
            cout << "Reached end of program" << endl;</pre>
           return 0;
                                            Explanation of output lines
         IntClass obj constructed!
                                            i1 gets constructed
         IntClass obj constructed!
                                            i2 gets constructed
         Done with declarations.
         Reached end of program
         IntClass obj destroyed - val was: 80
                                            i2 gets destroyed
         IntClass obj destroyed - val was: 50
                                            i1 gets destroyed
EECS
                                Andrew M Morgan
402
```



```
Fun With Ctors and Dtors, Cot'd
         class OuterClass
           public:
              int outInt;
              InnerClass inObj;
              InnerClass inObj2
              OuterClass()
                cout << "OuterClass obj created!" << endl;</pre>
              ~OuterClass()
                cout << "OuterClass obj dest. val:"</pre>
                      << outInt << endl;
         };
EECS
                               Andrew M Morgan
402
                Fun with Ctors and Dtors, cot'd
int main (void)
                                                   InnerClass obj created!
                                                   InnerClass obj created!
   InnerClass ic;
                                                   InnerClass obj created!
   OuterClass oc;
                                                   OuterClass obj created!
   cout << "Done with decls." << endl;</pre>
                                                   Done with decls.
   ic.inInt = 1;
                                                   Done with program
   oc.outInt = 2;
                                                   OuterClass obj dest. val:2
   oc.inObj.inInt = 3;
                                                   InnerClass obj dest. val: 4
                                                                               OC
   oc.inObj2.inInt = 4;
                                                   InnerClass obj dest. val: 3
   cout << "Done with program" << endl;</pre>
                                                   InnerClass obj dest. val: 1
   return 0;
}
                   Step through the code to determine what
                   order objects get created/destroyed in this
                   example.
EECS
```

Andrew M Morgan

402

```
Odd Output???
class MyClass
                                           From main:
                                              MyClass myObj;
  public:
    int val;
                                              printMyClass(myObj);
                                              myObj.val = 15;
    MyClass()
                                              printMyClass(myObj);
      val = 0;
      cout << "MyClass ctor" << endl;</pre>
                                                    MyClass ctor
                                                    MyClass val: 0
    ~MyClass()
                                                    MyClass dtor
      cout << "MyClass dtor" << endl;</pre>
                                                    MyClass val: 15
};
                                                    MyClass dtor
                                                   MyClass dtor
void printMyClass(const MyClass mc)
                                               Only 1 ctor message printed,
  cout << "MyClass val: "</pre>
                                               but 3 dtor messages printed!
       << mc.val << endl;
                                               What is happening?
EECS
                                Andrew M Morgan
703
```



Copy Constructor

三三〇8 402

- There is another special constructor, often called by default the copy constructor.
- Its signature looks like this:

```
ClassName (const ClassName &copy);
```

- The copy constructor is called when a new object is needed that is a copy of another object
- There is a "free copy ctor" that is automatically generated for each class you create
 - This means the function exists, even if the programmer didn't write it!
- The free copy constructor is automatically called when needed
 - i.e. when an object is passed-by-value into a function
 - The free copy ctor simply does a data member-by-member copy from the original object into the new object

402 402

Andrew M Morgan

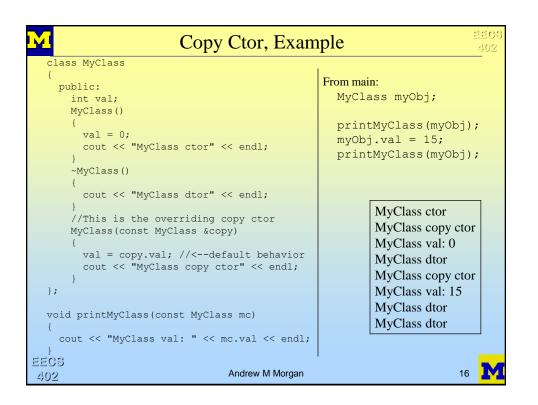
M

You can "override" the free copy ctor by implementing your own When you write your own copy ctor, it is called *in place of* the free copy ctor Therefore, if you want/need to add functionality to a copy ctor, you need to include the original functionality with the new The copy ctor is just another overloaded ctor Anytime any object is created one ctor will get called For value parameters, the new object is created and initialized using this version of the ctor

Andrew M Morgan

EECS

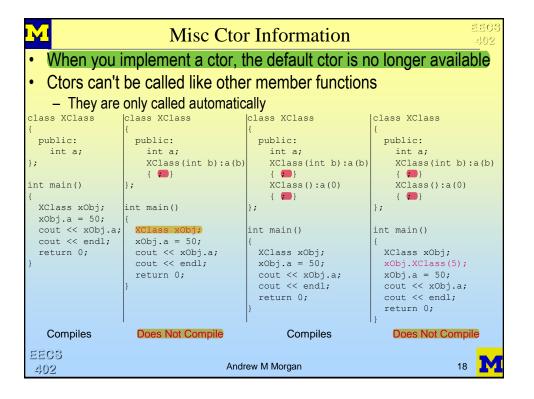
703



```
Constructor Initializers

    Initializing values in constructors is so common, a special syntax

  was developed
 Constructor initializers only work with constructors!
class InitClass
 public:
   InitClass():intVar(0), floatVar(0.0)
         //initializes intVar to 0 and
         //floatVar to 0.0
   InitClass(const int inI, const float inFf):intVar(inI), floatVar(inF)
         //initializes members to parameters
    //... More functions..
 private:
   int intVar;
   float floatVar;
EECS
                              Andrew M Morgan
703
```



```
Why Won't This Compile?
                                      class OutClass
class InClass
                                        private:
  private:
                                          int o;
    int i;
                                          InClass iObj;
  public:
                                        public:
    InClass(const int inI):i(inI)
                                          OutClass(const int in0):o(in0)
    void print() const
                                          void print() const
      cout << "i: " << i;
                                            cout << "o: " << o;
      cout << endl;
                                            cout << " iObj: ";
                                            iObj.print();
};
                                      };
//Continued next column
                                      int main()
                                        OutClass outClassObj (50);
                                        outClassObj.print();
                                        return 0;
EECS
                               Andrew M Morgan
703
```

M

Explanation Of Example Program

로로C8 402

- The iObj member of OutClass is an object
- In order to create an object, a ctor must be used
- The default ctor is used, since the use of another is not specified
 - The InClass has a ctor defined, though, therefore the default ctor is no longer available
- How about including a default ctor for InClass to fix it?
 - This would work however, what if you don't want a default ctor available to the user?
 - Perhaps you don't want to let the user to create an InClass object unless they provide a value to initialize the member to!
- Constructor initializers can be used to specify a ctor for initializing members

402

Andrew M Morgan

M

```
Updated Program For Compile
class InClass
                                          class OutClass
                                            private:
  private:
                                              int o;
InClass iObj;
    int i;
  public:
    InClass(const int inI):i(inI)
                                            public:
                                              OutClass(const int in0):o(in0), (iObj(0)
    void print() const
                                              void print() const
      cout << "i: " << i;
                                                cout << "o: " << o; cout << " iObj; ";
      cout << endl;
};
                                                iObj.print();
//Continued next column
                                          };
                                          int main()
                                            OutClass outClassObj(50);
This ctor initializer doesn't assign
                                            outClassObj.print();
                                            return 0;
iObj to 0 (check the types -
InClass is not an int). Instead, it
                                                                       o: 50 iObj: i: 0
specifies which version of the ctor
is used to create the iObj object.
EECS
                                     Andrew M Morgan
402
```