MIT OpenCourseWare
http://ocw.mit.edu

6.096 Introduction to C++
January (IAP) 2009

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.

Welcome to 6.096

Lecture 7
January 21, 2009

Constructors

- Objects need initialization to avoid the assignment of *junk* values.
- Constructor: just a member function.
- Initializes global variables.
- Called whenever a new object of this class is created.

Rules for making a constructor

- A constructor must have the same name as the class.
- No return type; not even void.
- No return statement.
- Never call a constructor manually. The execution process takes care of that.
- Never declare a constructor as virtual or static, const, volatile, or const volatile.
- References and pointers cannot be used on constructors and destructors because their addresses cannot be taken.

Example

```
class Rectangle
{
  int height;
  int width;
public:
     Rectangle();                    // with a constructor
   void printAns();
};

Rectangle::Rectangle()     // constructor
{
  height = 6;
  width = 6;
}
```

Default constructors

- A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values.

- No explicit constructor declaration => the compiler assumes the class to have a *default constructor* with no arguments.

Example

```
class iOffice
{
int a;
float b;
iOffice();
};

iOffice :: iOffice()          //default constructor
{
a = 100;
b = 5.5;
}
```

```
class Xamol {
public:

  // default constructor, no arguments
  Xamol();

  // constructor
  Xamol(int, int , int = 0);

  // copy constructor
  Xomal(const X&);
};

class Yamol {
public:

  // default constructor with one default argument
  Yamol( int = 0);

  // default argument copy constructor
  Yamol(const Y&, int = 0);
};
```

Copy constructor

- Used to copy an object to a newly created object.

- Different from assignment.

- If a copy constructor is not defined in a class, the compiler itself defines one.

- Used:
- – When an object is created from another object of the same type.
- – When an object is passed by value as a parameter to a function.
- – When an object is returned from a function.

Example

```
class MIT
{
    private:
            char *name;
    public:
            MIT()
            {
            name = new char[20];
            }


            MIT(const MIT &b)           //Copy constructor
            {
            name = new char[20];
            strcpy(name, b.name);
            }
};
```

Destructors

- Used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed

- Called for a class object when that object passes out of scope or is explicitly deleted.

- A destructor is a member function with the same name as its class prefixed by a ~ (tilde).

- Takes no arguments and has no return type.

- Its address cannot be taken.

- Cannot be declared const, volatile, const volatile or static.

- A destructor can be declared virtual or pure virtual.

Example

```
class Wheat
{
  int spoon;
  char fork;

  public:
   Wheat(int, char);          // Constructor for class Wheat

     ~Wheat();                // Default destructor for class Wheat

};

Wheat:: Wheat(int x, char y)
{
   spoon = x;
   fork = y;
}

Wheat:: ~Wheat()
{ }
```

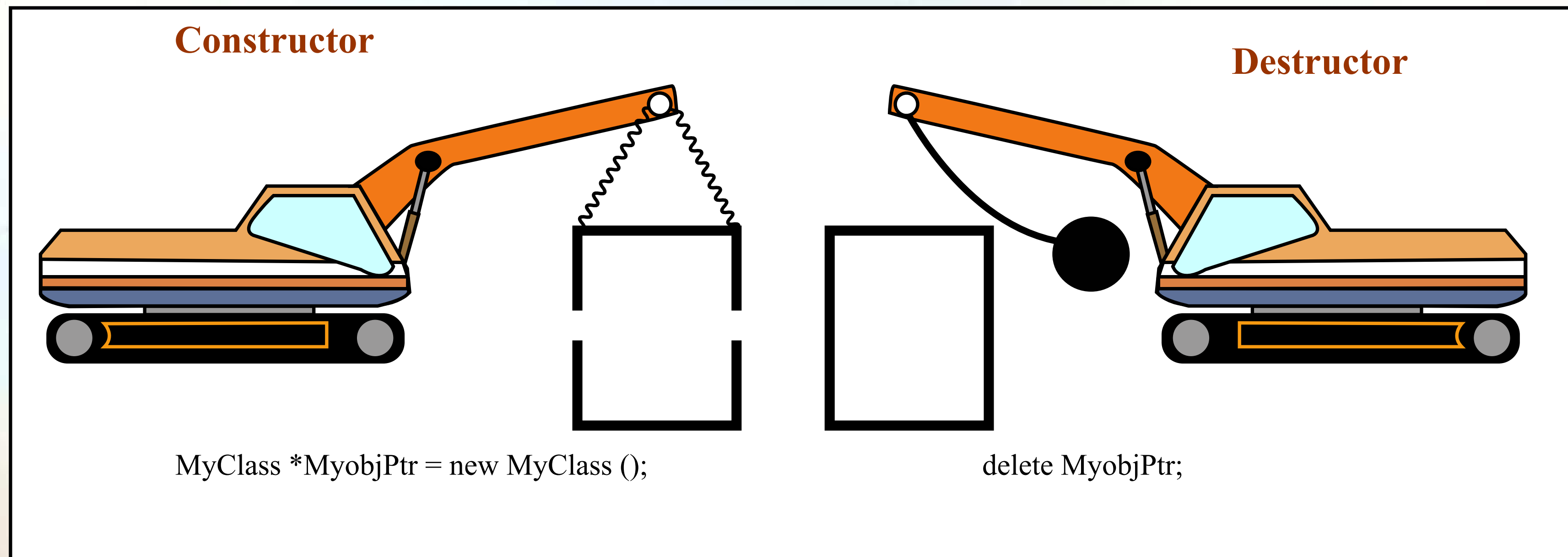

Figure by MIT OpenCourseWare.

The code for the actual construction or destruction of an object is added on by the compiler and you do not see it

Inheritance

- New classes called *derived classes* are created from existing classes called *base classes*

- When a class is inherited all the functions and data member are inherited, although not all of them will be accessible by the member functions of the derived class.

- Exceptions:
  - – The constructor and destructor of a base class are not inherited
  - – the assignment operator is not inherited
  - – the friend functions and friend classes of the base class are also not inherited.

Access specifiers

- Private: If a member or variables defined in a class is private, then they are accessible by members of the same class only and cannot be accessed from outside the class.

- Public members and variables are accessible from outside the class.

- Protected access specifier is a stage between private and public. If a member functions or variables defined in a class are protected, then they cannot be accessed from outside the class but can be accessed from the derived class.

Access	public	protected	Private
members of the same class	yes	yes	yes
members of derived classes	yes	yes	no
not members	yes	no	no

Implementing inheritance

- class <*derived_classname*> : <*access specifier*> <*base_classname*>
- {
  - …
- };

- class Daughter : public Mother
- {

....

};

```
class MIT
{
public:
      MIT() { x=0; }
      void func(int n1) { x = n1*5; }
      void output() { cout << x << '\n'; }

private:
      int x;
};

class IAP : public MIT
{
    public:
      IAP() { s1=0; }

      void func1(int n1)
      {
          s1=n1*10;
      }

      void output()
      {
          MIT::output();
        cout << s1 << '\n';
      }

private:
      int s1;
};

int main()
{
IAP obj;
obj.func(10);
obj.output();
obj.func1(20);
obj.output();
}
```

0 50
50 200

Multiple inheritance

- Inheriting from more than one class

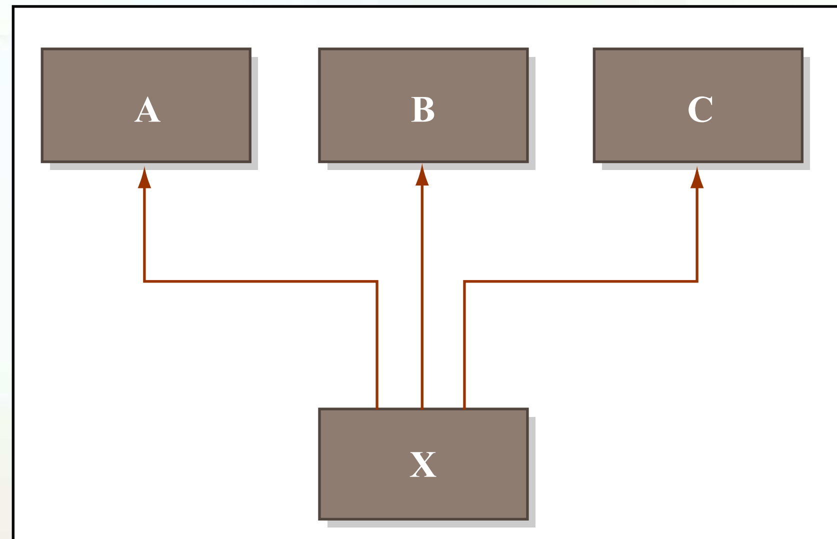

Figure by MIT OpenCourseWare.

- Separate the different base classes with commas in the derived class declaration

- class Daughter: public Mother, public Father;