

## More about classes: Constructors

Wagner Truppel  
Lecturer, Dept. of Computer  
Science & Engineering  
UC Riverside

[wagner@cs.ucr.edu](mailto:wagner@cs.ucr.edu)  
<http://www.cs.ucr.edu/~wagner>

<http://www.cs.ucr.edu/cs12>

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 1

---

---

---

---

---

---

---

---

## Recall this example...

```
class Point
{
    private:
        int x;
        int y;
    public:
        Point(int ax, int ay); // this is a constructor
        int getX(); // this is an accessor function
        int getY(); // this is an accessor function
        void setX(int ax); // this is a mutator function
        void setY(int ay); // this is a mutator function
};
```

Point
- int x
- int y
+ Point(int ax, int ay)
+ int getX()
+ int getY()
+ void setX(int ax)
+ void setY(int ay)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 2

---

---

---

---

---

---

---

---

## Constructors

Point
- int x
- int y
+ Point(int ax, int ay)
+ int getX()
+ int getY()
+ void setX(int ax)
+ void setY(int ay)

- Special kind of member function
  - ◆ Used to initialize member variables
  - ◆ More generally, used to initialize the **state** of the object being created
  - ◆ Automatically invoked when declaring a class variable (an object of the class)
- Must have the same name as the class they're defined for
- No return type (**not** even *void*)
- You **can** have more than one
- **Not** invoked like other functions (more on this later)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 3

---

---

---

---

---

---

---

---

## Declaring constructors...

```

class Point
{
    private:
        int x;
        int y;
    public:
        Point(int ax, int ay); // this is a constructor
        int getX(); // this is an accessor function
        int getY(); // this is an accessor function
        void setX(int ax); // this is a mutator function
        void setY(int ay); // this is a mutator function
};
        
```

Point
- int x
- int y
+ Point(int ax, int ay)
+ int getX()
+ int getY()
+ void setX(int ax)
+ void setY(int ay)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 4

---

---

---

---

---

---

---

---

---

---

## Declaring constructors...

Display 7.1,  
page 262

```

// File: classWithConstructors.cpp
#include <iostream> // for exit
using namespace std;

4 class DogPup {
5     public:
6         DogPup(int monthVal, int dayVal) {
7             //initialize the month and day to arguments.
8         }
9         DogPup(int monthVal) {
10            //initialize the month and day to arguments.
11        }
12        DogPup() { // default constructor
13            //initialize the date to January 1.
14        }
15        void input();
16        void output();
17        int getMonth();
18        int getDay();
19        void testDate();
20    };
21
22 int main()
23 {
24     DogPup dave(2, 20), dora(3, dave);
25     cout << "initialized dave:\n";
26     dave.output(); cout << endl;
27     dora.output(); cout << endl;
28     dave.output(); cout << endl;
29     dora.output(); cout << endl;
30     return 0;
31 }
32
33 DogPup::DogPup(int monthVal, int dayVal)
34 : month(monthVal), day(dayVal)
35 {
36     testDate();
37 }
        
```

This defines DogPup as an improved version of the class DogPup given in Display 7.0.

This causes a call to the default constructor. Notice that there are no parentheses.

an explicit call to the constructor DogPup::DogPup()

Copyright © 2002 Pearson Education, Inc. CS 12: Intro. Computer Science II • Lecture 8 5

---

---

---

---

---

---

---

---

---

---

## Defining constructors...

- Somewhere on the same file, after the class declaration
- 2 ways to declare a constructor:
 

```

Point::Point(int ax, int ay)
{
    x = ax;
    y = ay;
}
// we could now verify the input

Point::Point(int ax, int ay) : x(ax), y(ay)
{
    // this body does not have to be empty !
    // we can use it to verify the input
}
            
```
- The second one is considered preferable (it's more efficient)

Point
- int x
- int y
+ Point(int ax, int ay)
+ int getX()
+ int getY()
+ void setX(int ax)
+ void setY(int ay)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 6

---

---

---

---

---

---

---

---

---

---



### Invoking constructors...

- Object declaration is similar
  - ◆ *Point p*; declares *p* as an object of class *Point*, allocates space for it, **but also invokes a constructor**
  - ◆ But which constructor? We only defined one and it takes arguments
- The invoked constructor is **Point()**
- But we didn't define it, so the compiler complains

Point
- int x
- int y
+ Point(int ax, int ay)
+ int getX()
+ int getY()
+ void setX(int ax)
+ void setY(int ay)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 10

---

---

---

---

---

---

---

---

### Invoking constructors...

- The **no-argument** constructor is called the **default** constructor
- If you don't define any constructor for your class, the compiler automatically creates one for you - the default constructor
- But if you **do** define *any* constructor, the compiler will **not** create the default for you
- In such a case, declaring an object as in *Point p*; is an **error** (unless one of the constructors you defined is the default)

Point
- int x
- int y
+ Point(int ax, int ay)
+ int getX()
+ int getY()
+ void setX(int ax)
+ void setY(int ay)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 11

---

---

---

---

---

---

---

---

### Invoking constructors...

- What if you do define the no-argument constructor? How do you invoke it?  
 DayOfYear today;  
 DayOfYear today = DayOfYear();
- Compare with  
 DayOfYear today(2, 11);  
 DayOfYear today = DayOfYear(2, 11);
- For a good example that puts it all together, look at the **BankAccount** class on pp. 268-273 of the textbook

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 12

---

---

---

---

---

---

---

---

## Passing objects to functions

- Objects are typically “heavy” (have lots of internal stuff)
- Better never to pass them by value but, instead, by reference (to avoid copying)
- But what if you don’t want to change the object’s internal state? Use **const**  
 void drawRect(**const** Rectangle& rect, **const** Color& color)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 13

---

---

---

---

---

---

---

---

## A concrete example

```

class Rectangle
{
private:
    Point top_left;
    Point bottom_right;
public:
    Rectangle(Point topL, Point botR);
    Point getTopLeft() const;
    Point getBottomRight() const;
    void setTopLeft(Point p);
    void setBottomRight(Point p);
    int getWidth() const;
    int getHeight() const;
};
    
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 14

---

---

---

---

---

---

---

---

## A concrete example

```

Rectangle::Rectangle(Point topL, Point botR) : top_left(topL),
bottom_right(botR) {}

Point Rectangle::getTopLeft() const
{ return top_left; }

Point Rectangle::getBottomRight() const
{ return bottom_right; }

void Rectangle::setTopLeft(Point p)
{ top_left = p; }

void Rectangle::setBottomRight(Point p)
{ bottom_right = p; }

int Rectangle::getWidth() const // assuming x grows to the right
{ return (bottom_right.x - top_left.x); }

int Rectangle::getHeight() const // assuming y grows downwards
{ return (bottom_right.y - top_left.y); }
    
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 15

---

---

---

---

---

---

---

---

## Another use for const

- Sometimes we don't want a member function of a class to change the state of the object on which it's invoked

```

Rectangle r =
  Rectangle(Point(2, 3), Point(10, 17));
cout << "Width = " << r.getWidth() << endl;
    
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 16

---

---

---

---

---

---

---

---

## Static members

- Sometimes we want to have a member variable which is shared by all objects of a given class
- Example: a count of how many objects have been created
- Such variables are not tied to any one object in particular
- Member variables like this are called **static member variables**
- They are a sort of global variable
- They must be **initialized only once**
- They must be initialized **outside the class** they're defined in

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 17

---

---

---

---

---

---

---

---

## Static member variables

```

class Connection
{
  private:
    // private members
  public:
    Connection(); // default constructor
    static int connCount; // public just as an example
    // other public members
};

int Connection::connCount = 0;

Connection::Connection()
{
  if (connCount > 10)
    ; // should raise an exception (an error)
  connCount++;
}
    
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 18

---

---

---

---

---

---

---

---

### Static member variables

- How do you access them?
- Since they're not bound to any one object, you could access them from any object of its class:  
`Connection conn1; // calls the default constructor`  
`cout << conn1.connCount;`
- The preferred way, however, is to use the class name:  
`cout << Connection::connCount;`
- In this example, this is possible because `connCount` was defined as **public**. What if it was a private variable?

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 19

---

---

---

---

---

---

---

---

### Static member variables

```
class Connection
{
private:
    static int connCount;
    // other private members
public:
    Connection(); // default constructor
    static int getNumConns();
    // other public members
};

int Connection::connCount = 0;

Connection::Connection()
{
    if (connCount > 10)
        ; // should raise an exception (an error)
    connCount++;
}

int Connection::getNumConns()
{ return connCount; }
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 20

---

---

---

---

---

---

---

---

### Static member variables

- How do you access a private static member variable?
- You need an **accessor** member function
- But since the variable is static, the accessor function must also be static  
`cout << Connection::getNumConns();`
- You cannot access object-specific info from within a static function

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 21

---

---

---

---

---

---

---

---

**A bit of perspective**

- What do we gain by using well-defined classes?
  - ◆ Self-contained components
  - ◆ Easier to develop
  - ◆ Easier to debug
  - ◆ Easier to maintain
  - ◆ Re-usable
  - ◆ Are good representations of the world

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 8 22

---

---

---

---

---

---

---

---