Chapter 13

Introduction to Classes, Cont'd

NOTES

- **NO LAB this week**
- **You should be working on program 5**

WARM UP

- What three files are usually used to separate definition from implementation from user/client of the class?
  - Header (*classname*.h)
  - Implementation (*classname*.cpp)
  - Code using the class (*xxxclient*.cpp)
- What is “stale” data?
  - When data member value depends on other data values and is <u>not</u> updated
- What is an inline function? Why use it?
  - Improves program performance
  - Inline function code is placed “inline” in place of a call statement

13.7

CONSTRUCTORS

CONSTRUCTOR

What can you tell me about this object (instance of Cat class) when it is created.

```
Cat fluffy;
```

- It is allocated statically.
- It is uninitialized, i.e., age of fluffy is “garbage”.
- What is the user obligated to do before using object fluffy?

CONSTRUCTOR

A ***member function*** that is automatically called when an object is created (i.e., "constructed")

```
Cat fluffy;
```

Purpose is to set up the object, i.e., help with the house keeping chores…

Constructor function name is the same as the class name

Has no return type (it is not void)

Contents of `Rectangle.h` (Version 3)

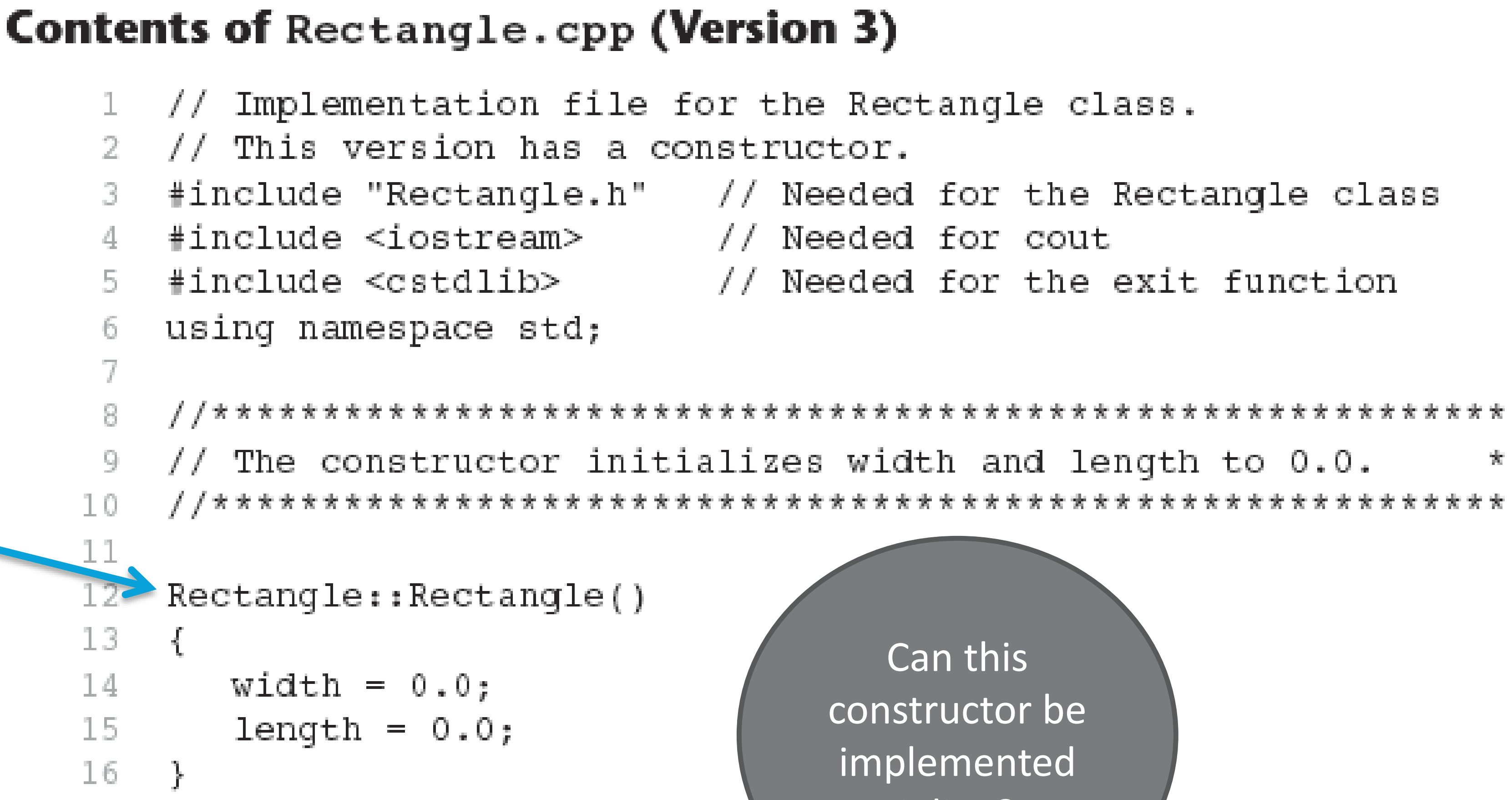
```
// Specification file for the Rectangle class
// This version has a constructor.
#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle
{
   private:
      double width;
      double length;
   public:
      Rectangle();                  // Constructor
      void setWidth(double);
      void setLength(double);

      double getWidth() const
         { return width; }

      double getLength() const
         { return length; }

      double getArea() const
         { return width * length; }
};
#endif
```



Contents of Rectangle.cpp (Version 3)

```
// Implementation file for the Rectangle class.
// This version has a constructor.
#include "Rectangle.h"   // Needed for the Rectangle class
#include <iostream>      // Needed for cout
#include <cstdlib>       // Needed for the exit function
using namespace std;

//***********************************************************
// The constructor initializes width and length to 0.0.     *
//***********************************************************

Rectangle::Rectangle()
{
   width = 0.0;
   length = 0.0;
}
```

Can this constructor be implemented inline?

Continues...

DEFAULT CONSTRUCTORS

A special kind of constructor is the default constructor.

It takes no arguments.

If you write a class with no constructor at all, C++ will write a default constructor for you, one that does nothing.

A simple instantiation of a class (with no arguments) calls the default constructor:

```
Rectangle my_room;    // default constructor is called
Cat       fluffy;     // default constructor is called
```

IN CLASS EXERCISE - CONSTRUCTORS

Add the default constructor to the Cat class we implemented earlier.

IN CLASS EXERCISE - CONSTRUCTORS

```
// Class definition (specification): Cat.h
#ifndef CAT_H
#define CAT_H

class Cat
{
private:
        // Data
        int age; // in years
public:
        // Default Constructor, implemented out-of-line
        Cat();

        // rest of the class
};
```

```
// cat.cpp
#include "cat.h"
Cat::Cat()
{
  age=0;
}
```

IN CLASS EXERCISE - CONSTRUCTORS

```
// Class definition (specification): Cat.h
#ifndef CAT_H
#define CAT_H

class Cat
{
private:
         // Data
         int age; // in years
public:
         // Default Constructor, implemented in-line
         Cat() { age = 0; }

         // rest of the class
};
```

13.8

PASSING ARGUMENTS TO CONSTRUCTORS

ADDITIONAL CONSTRUCTORS

- A class can have more than one constructor
- All constructors have the same name but ***different parameters***
- Which constructor is automatically executed?
  - It depends on how the object is created…

```
Rectangle my_room;
Rectangle your_room(12.5, 15.0);
```

PASSING DATA TO CONSTRUCTORS

To create a constructor that accepts parameters:

- provide parameters in the prototype:

```
Rectangle(double, double);
```

- Use parameters in the definition:

```
Rectangle::Rectangle(double w, double l)
{
   width = w;          // private data member width initialized
   length = l;         // private data member length initialized
}
```

IN CLASS EXERCISE - CONSTRUCTORS

```
// Class definition (specification): Cat.h

#ifndef CAT_H
#define CAT_H

class Cat
{
private:
         // Data
         int age; // in years
public:
         // Default Constructor, implemented in-line
         Cat() { age = 0; }
         // Parameterized constructor, implemented in-line
         Cat(int a) { age = a; }
```

PASSING ARGUMENTS TO CONSTRUCTORS

You can then pass arguments to the constructor when you create an object:

```
int main()
  {
    Rectangle my_room;                 // default constructor called
    Rectangle your_room(12.5, 14.0);  // other constructor called


    return 0;
  }
```

```
int main()
{
     Cat fluffy;
     Cat whiskers(2);


     return 0;
}
```

MORE ABOUT DEFAULT CONSTRUCTORS

If ***all*** of a constructor's parameters have **default values**, then it is considered the default constructor.

For example:

```
Rectangle(double=0, double=0);
```

Creating an object and passing no arguments will cause this constructor to be called:

```
Rectangle room;
```

class can only have **one** default constructor!

MORE ABOUT DEFAULT CONSTRUCTORS

```
// if we have the following in class definition

class Rectangle{
private:
      double width, height;
public:
      Rectangle();        // default constructor
      Rectangle(double=0, double=0);

      // other methods
};
```

compiler
will not
allow this!

CLASSES WITH NO DEFAULT CONSTRUCTOR

When all of a class's constructors require arguments (without default values), then the class has NO **default constructor.**

When this is the case, you **must** pass the required arguments to the constructor when creating an object.

When this is the case, the compiler will **NOT** create a default constructor for you!

```
Rectangle my_room;              // not allowed
Rectangle my_room(10, 20);      // allowed
```

SUMMARY OF CONSTRUCTOR INFO

Constructors in class definition	What happens	Client code
No constructors	C++ provides one **BUT** it does nothing; data still uninitialized	Cat fluffy;

SUMMARY OF CONSTRUCTOR INFO

Constructors in class definition	What happens	Client code
No constructors	C++ provides one **BUT** it does nothing; data still uninitialized	Cat fluffy;
(1) Default constructor only: `Cat(){age=0;}`	OK	Cat fluffy;

SUMMARY OF CONSTRUCTOR INFO

Constructors in class definition	What happens	Client code
No constructors	C++ provides one **BUT** it does nothing; data still uninitialized	Cat fluffy;
(1) Default constructor only: `Cat(){age=0;}`	OK	Cat fluffy;

(1) Default constructor:
(2) Parameterized constructor: `Cat(int a){age=a;}`
	OK	Cat fluffy;
		Cat wiskers(1);

SUMMARY OF CONSTRUCTOR INFO

Constructors in class definition	What happens	Client code
No constructors	C++ provides one **BUT** it does nothing; data still uninitialized	Cat fluffy;
(1) Default constructor only: `Cat(){age=0;}`	OK	Cat fluffy;

(1) Default constructor:
(2) Parameterized constructor: `Cat(int a){age=a;}`
	OK	Cat fluffy;
		Cat wiskers(1);

(1) Default constructor: `Cat(){age=0;}` (2) Parameterized constructor with default values for all parameters: `Cat(int a=0){age=a;}`	Syntax error – cannot have two default constructors	

SUMMARY OF CONSTRUCTOR INFO

Constructors in class definition	What happens	Client code
No constructors	C++ provides one **BUT** it does nothing; data still uninitialized	Cat fluffy;
(1) Default constructor only: `Cat(){age=0;}`	OK	Cat fluffy;

(1) Default constructor:
(2) Parameterized constructor: `Cat(int a){age=a;}`
	OK	Cat fluffy;
		Cat wiskers(1);

(1) Default constructor: `Cat(){age=0;}` (2) Parameterized constructor with default values for all parameters: `Cat(int a=0){age=a;}`	Syntax error – cannot have two default constructors	

(1) Parameterized constructor only with no default values: `Cat(int a){age=a;}`	C++ will **not** provide the default constructor; user **must** provide arguments when creating objects	Cat fluffy; // error
		Cat fluffy(0);

SUMMARY OF CONSTRUCTOR INFO

Constructors in class definition	What happens	Client code
No constructors	C++ provides one **BUT** it does nothing; data still uninitialized	Cat fluffy;
(1) Default constructor only: `Cat(){age=0;}`	OK	Cat fluffy;

(1) Default constructor:
(2) Parameterized constructor: `Cat(int a){age=a;}`
	OK	Cat fluffy;
		Cat wiskers(1);

(1) Default constructor: `Cat(){age=0;}` (2) Parameterized constructor with default values for all parameters: `Cat(int a=0){age=a;}`	Syntax error – cannot have two default constructors	

(1) Parameterized constructor only with no default values: `Cat(int a){age=a;}`	C++ will **not** provide the default constructor; user **must** provide arguments when creating objects	Cat fluffy; // error
		Cat fluffy(0);

(1) Parameterized constructor only with default values for all parameters: `Cat(int a=0){age=a;}`	OK	Cat fluffy;
		Cat whiskers(1);

13.9

DESTRUCTORS

DESTRUCTORS

A ***member function*** automatically called when an object is **destroyed**

Destructor name is ~classname(), *e.g.*, **`~Rectangle(), ~Cat()`**

Has NO return type; takes NO arguments

Only **one** destructor per class is allowed!

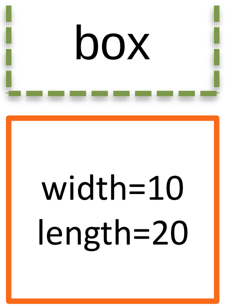
Performs "house-keeping" duties… on object destruction

If constructor allocates dynamic memory, destructor should release it!

CONSTRUCTORS, DESTRUCTORS, AND STATICALLY ALLOCATED OBJECTS

When an object is statically allocated, appropriate constructor executes:

```
   Rectangle box(10, 20);
```



Object box is destroyed automatically when program/function ends and the class destructor executes but has nothing to do.

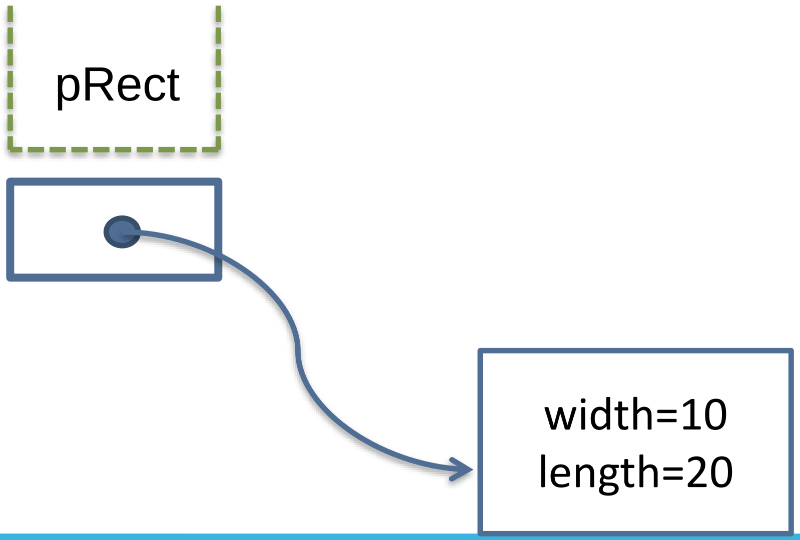
CONSTRUCTORS, DESTRUCTORS, AND DYNAMICALLY ALLOCATED OBJECTS

When an object is dynamically allocated, appropriate constructor executes:

```
Rectangle* pRect = new Rectangle(10, 20);
```

Object is destroyed on **delete** instruction, its destructor executes.

```
delete pRect;
pRect = nullptr;
```



13.10

OVERLOADING CONSTRUCTORS

WHAT IS OVERLOADING?

Not unique to OOP (Object Oriented Programing)…

Two or more functions that have the same name but differ in their **parameters**…

```
double CalcWeeklyPay(int hours, double payRate)
{
    return hours * payRate;
}

double CalcWeeklyPay(double annualSalary)
{
    return annualSalary / WEEKS_IN_YEAR;
}
```

OVERLOADING CONSTRUCTORS

A class can have more than one constructor

Overloaded constructors (i.e., parameterized constructors) in a class must have ***different*** parameter lists:

```
Rectangle();                    // default constructor
Rectangle(double);              // overloaded constructor
Rectangle(double, double);      // overloaded constructor
```

```
// This class has overloaded constructors.
#ifndef INVENTORYITEM_H
#define INVENTORYITEM_H
#include <string>
using namespace std;

class InventoryItem
{
private:
   string description; // The item description
   double cost;        // The item cost
   int units;          // Number of units on hand
public:
   // Constructor #1
   InventoryItem()
      { // Initialize description, cost, and units.
        description = "";
        cost = 0.0;
        units = 0; }

   // Constructor #2
   InventoryItem(string desc)
      { // Assign the value to description.
        description = desc;

        // Initialize cost and units.
        cost = 0.0;
        units = 0; }
```

Continues...

```

   // Constructor #3
   InventoryItem(string desc, double c, int u)
     { // Assign values to description, cost, and units.
       description = desc;
       cost = c;
       units = u; }

   // Mutator functions
   void setDescription(string d)
      { description = d; }

   void setCost(double c)
      { cost = c; }

   void setUnits(int u)
      { units = u; }

   // Accessor functions
   string getDescription() const
      { return description; }

   double getCost() const
      { return cost; }

   int getUnits() const
      { return units; }
};
#endif
```

ONLY **ONE** *DEFAULT* CTOR AND ONE DTOR

DO NOT provide more than one default constructor for a class.

Provide either one that **takes no arguments** OR one that has **default arguments for all parameters** but not both.

```
Square();
Square(int=0);  // considered a default constructor
```

Since a destructor takes no arguments, there can only be one destructor for a class, i.e., we can’t overload

IN CLASS EXERCISE

```
class bagType
{
public:
    _______________________________________  // default constructor
    _______________________________________  // overloaded constructor
    void set(string, double, double, double, double);
    void print() const;
    string getStyle() const;
    double getPrice() const;
    void get(string&, double&, double&, double&, double&) const;

private:
    string style;
    double length, width, height, price;
};
```

IN CLASS EXERCISE

How many members does class `bagType` have?

10 (5 data, 5 functions)

How many private members does class `bagType` have?

5 (all of the data members)

How many accessor functions does class `bagType have?`

4 (all the ones with "const")

IN CLASS EXERCISE

Write the default constructor for this class. Assume it will be implemented as an out-of-line function.

```
bagType::bagType()
{
    style="";
    length=0;
    width=0;
    height=0;
    price=0.0;
}
```

IN CLASS EXERCISE

Write an overloaded (parameterized) constructor to initialize all private data members. Assume it will be implemented as an out-of-line function.

```
bagType::bagType(string s, double l, double w,
                 double h, double p)
{
    style=s;
    length=l;
    width=w;
    height=h;
    price=p;
}
```

IN CLASS EXERCISE

Assume variable declaration in client code's main()…

```
bagType purse;
```

Which constructor is used to initialize object purse?

the default constructor

Write code to declare an object of type bagType, called *knapsack*, and use the values "backpack", 18, 6, 24, and 100 to initialize the object's data members.

```
bagType knapsack("backpack", 18, 6, 24, 100);
```

Which constructor is used to initialize this object?

the overloaded (parameterized) constructor

IN CLASS EXERCISE

```
class bagType
{
public:
   bagType();
   bagType(string, double, double, double, double);
   void set(string, double, double, double, double);
   void print() const;
   string getStyle() const;
   double getPrice() const;
   void get(string&, double&, double&, double&, double&);
private:
   string style;
   double l, w, h, price;
};
```

IN CLASS EXERCISE

Write a prototype for the *destructor* of class `bagType`

```
~bagType();
```