Foundation of Computer Science: Class

Kafi Rahman

Assistant Professor

Computer Science

Truman State University

13.5

Separating Specification from Implementation

Separating Specification from Implementation

- Place class declaration in a header file that serves as the class specification file.
  - Name the file ClassName.h, for example, Rectangle.h
- Place member function definitions in ClassName.cpp, for example, Rectangle.cpp
  - File should #include the class specification file
- Programs that use the class must #include the class specification file. And that's it.

13.6

Inline Member Functions

Inline Member Functions

- Member functions can be defined
  - inline: in class declaration
  - after the class declaration

- Inline appropriate for short function bodies:

  -
    ```
    int getWidth() const
       { return width; }
    ```

Rectangle Class with Inline Member Functions

```cpp
// Specification file for the Rectangle class
// This version uses some inline member functions.
#ifndef RECTANGLE_H
#define RECTANGLE_H

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

    double getWidth() const
    { return width; }

    double getLength() const
    { return length; }

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

Tradeoffs – Inline vs. Regular Member Functions

- Regular functions – when called, compiler stores return address of call, allocates memory for local variables, etc.

- Code for an inline function is copied into program in place of call
  - results in larger executable program, but
  - no function call overhead, hence,
  - faster execution

13.7

Constructors

Constructors

- Member function that is automatically called when an object is created
- Purpose is to initialize an object
- Constructor function name is the class name
- Has no return type (they return the object)

Constructors (cont)

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
```

Constructors (cont)

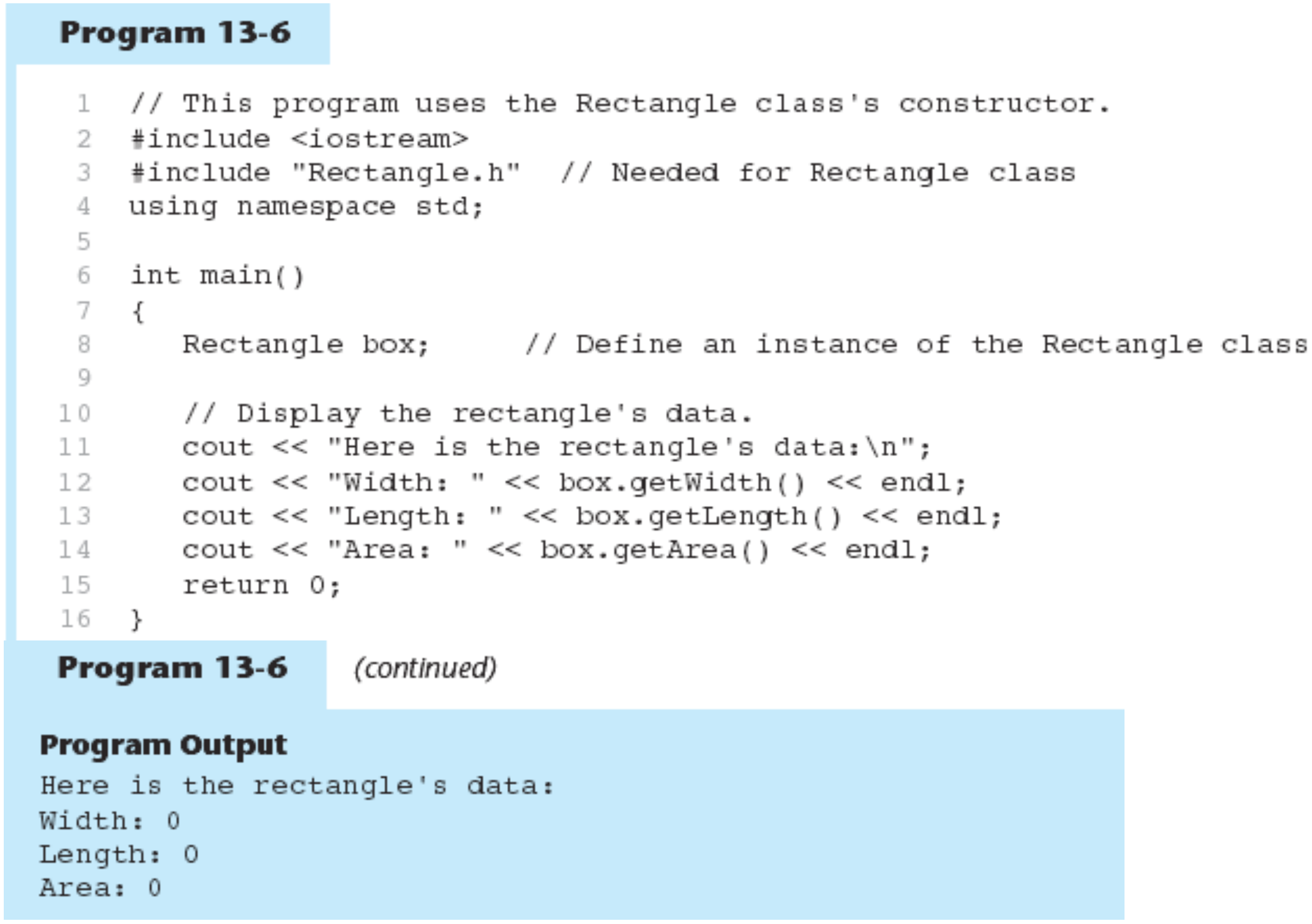
Contents of `Rectangle.cpp` (Version 3)

```
// 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;
}
```

Constructors (cont)

```cpp

//***********************************************************
// setWidth sets the value of the member variable width.    *
//***********************************************************

void Rectangle::setWidth(double w)
{
   if (w >= 0)
      width = w;
   else
   {
      cout << "Invalid width\n";
      exit(EXIT_FAILURE);
   }
}

//***********************************************************
// setLength sets the value of the member variable length.  *
//***********************************************************

void Rectangle::setLength(double len)
{
   if (len >= 0)
      length = len;
   else
   {
      cout << "Invalid length\n";
      exit(EXIT_FAILURE);
   }
}
```

Constructors (cont)



Program 13-6

```cpp
// This program uses the Rectangle class's constructor.
#include <iostream>
#include "Rectangle.h"  // Needed for Rectangle class
using namespace std;

int main()
{
   Rectangle box;     // Define an instance of the Rectangle class

   // Display the rectangle's data.
   cout << "Here is the rectangle's data:\n";
   cout << "Width: " << box.getWidth() << endl;
   cout << "Length: " << box.getLength() << endl;
   cout << "Area: " << box.getArea() << endl;
   return 0;
}
```

Program 13-6 *(continued)*

Program Output

```
Here is the rectangle's data:
Width: 0
Length: 0
Area: 0
```

Default Constructors

- A default constructor is a constructor that takes no arguments.
- If you write a class with no constructor at all, C++ will create a default constructor for you (not visible), one that does nothing.
- A simple instantiation of a class (with no arguments) calls the default constructor:
    - Rectangle r;

13.8

Passing Arguments to Constructors

Passing Arguments to Constructors

- To create a constructor that takes arguments:
  - indicate parameters in prototype:

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

  - Use parameters in the definition:

    ```
    Rectangle::Rectangle(double w, double len)
    {
        width = w;
        length = len;
    }
    ```

Passing Arguments to Constructors

- Now, we can pass arguments to the constructor when we create an object:
  - Rectangle r(10, 5);

Default Arguments

- We can specify default values for the function parameters
- The values should be provided from right to left
  - i.e., it is possible that parameter on the left will not have default value
  - the vice-versa is not supported in C++

```cpp
// y and z have default values
void abc(int x, int y=100, int z=50)
{
  cout<< x << " "
      << y << " "
      << z << endl;
}

// driver function
int main()
{
  // the default values will be used
  // for the other two parameters
  abc(10);
  return 0;
}
```