03: Constructors and Destructors

Programming Technique II
(SCSJ1023)

Adapted from Tony Gaddis and Barret Krupnow (2016), Starting out with C++: From Control Structures through Objects

Constructors

Constructors

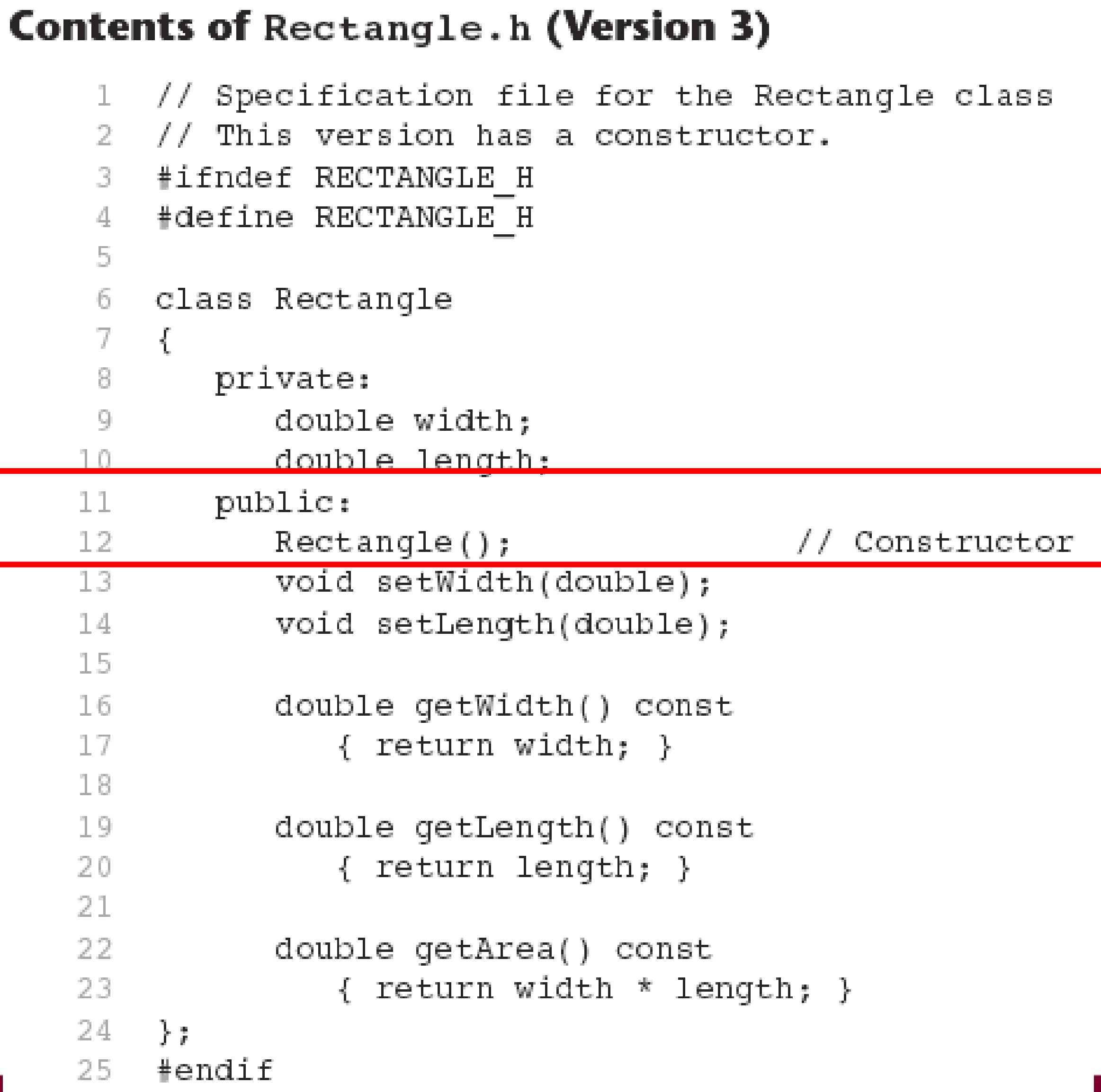
- Member function that is automatically called when an object is created.

- Purpose is to construct an object.

- Constructor function name is class name.

- Has no return type.

Example 1: Constructors

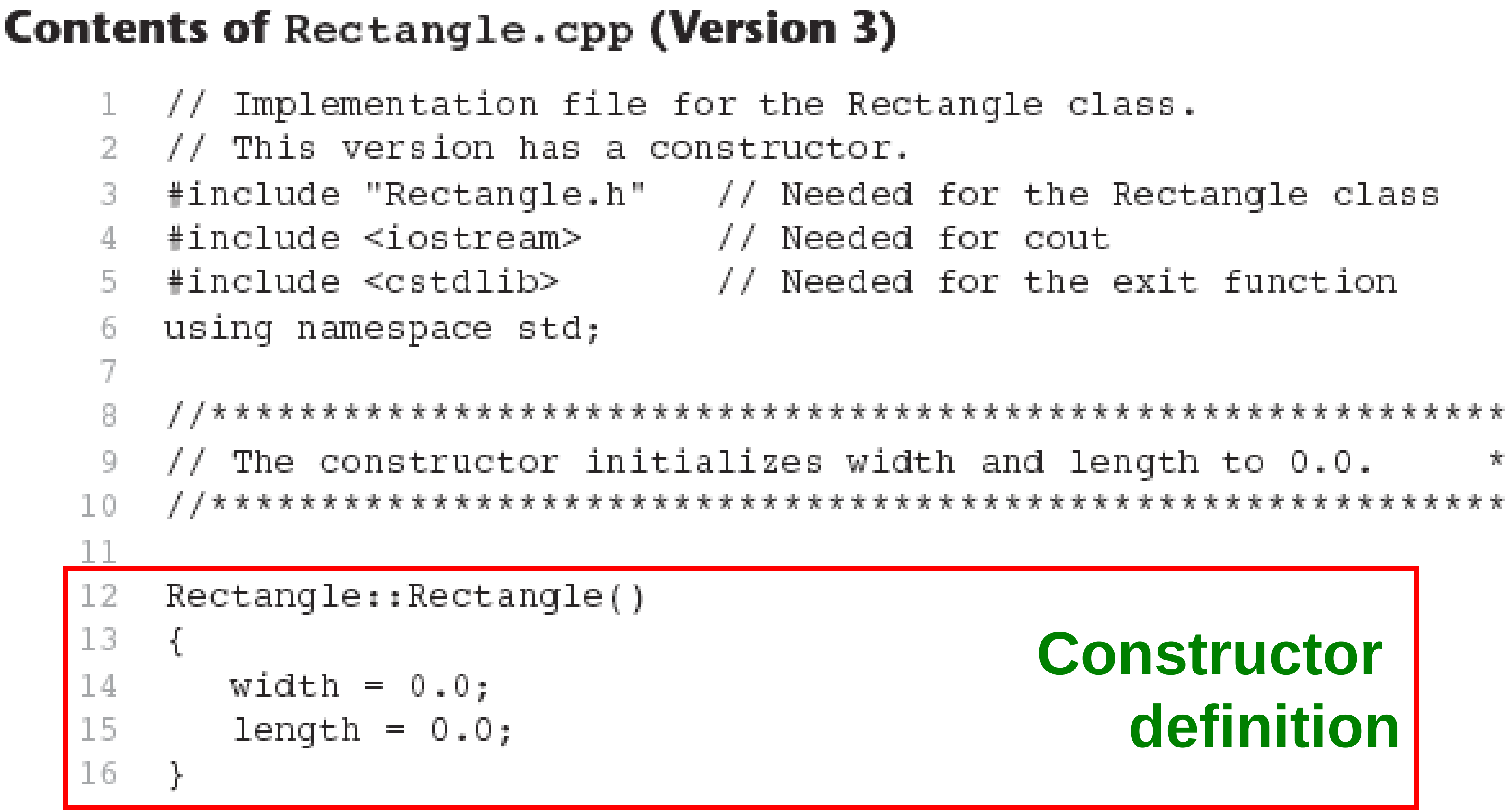


Contents of `Rectangle.h` (Version 3)

```
// 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
```

Constructor in class definition

Example 1: Constructors (cont’)



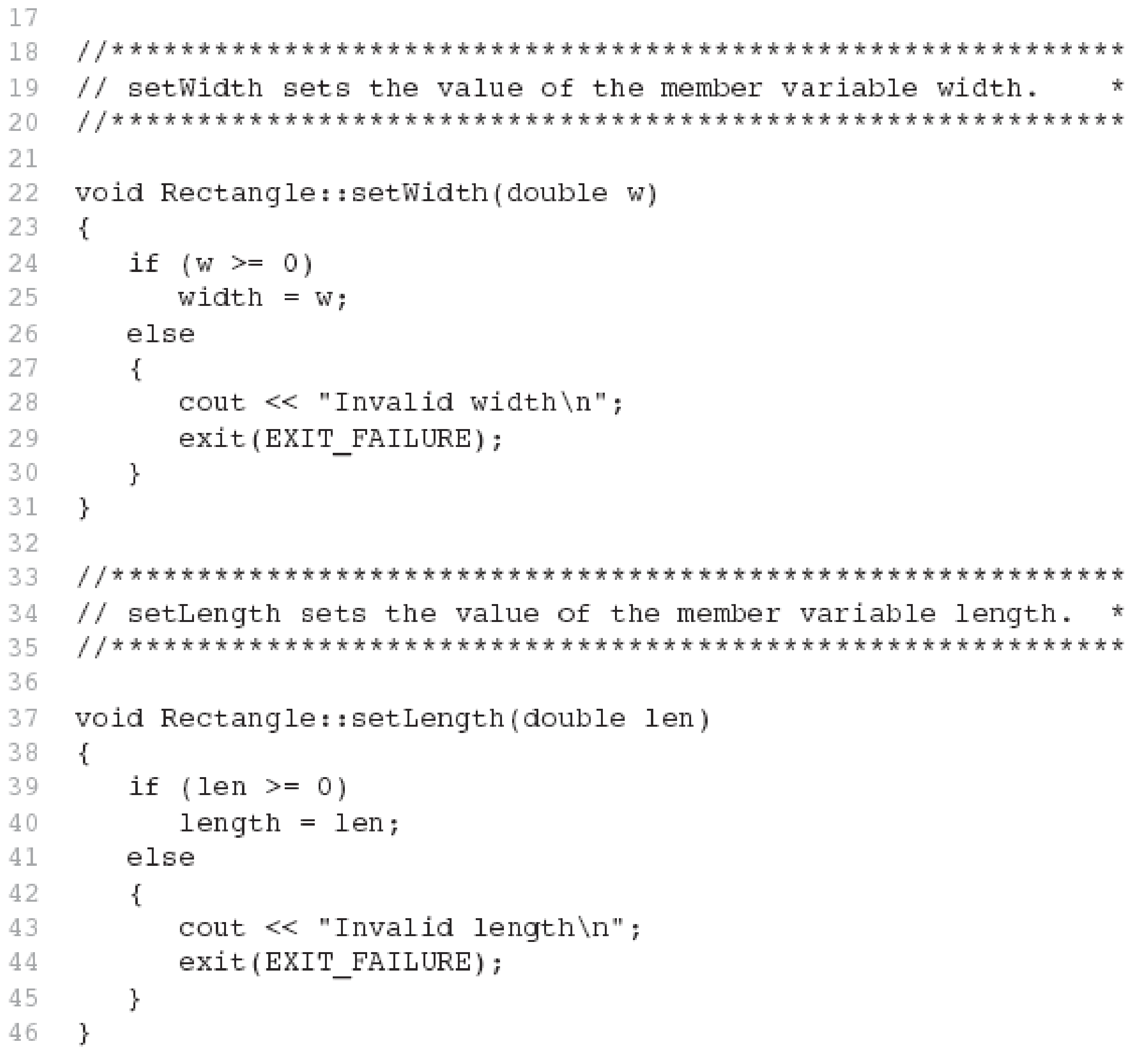
Contents of `Rectangle.cpp` (Version 3)

```
// 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;
}
```

Constructor definition

Example 1: Constructors (cont')

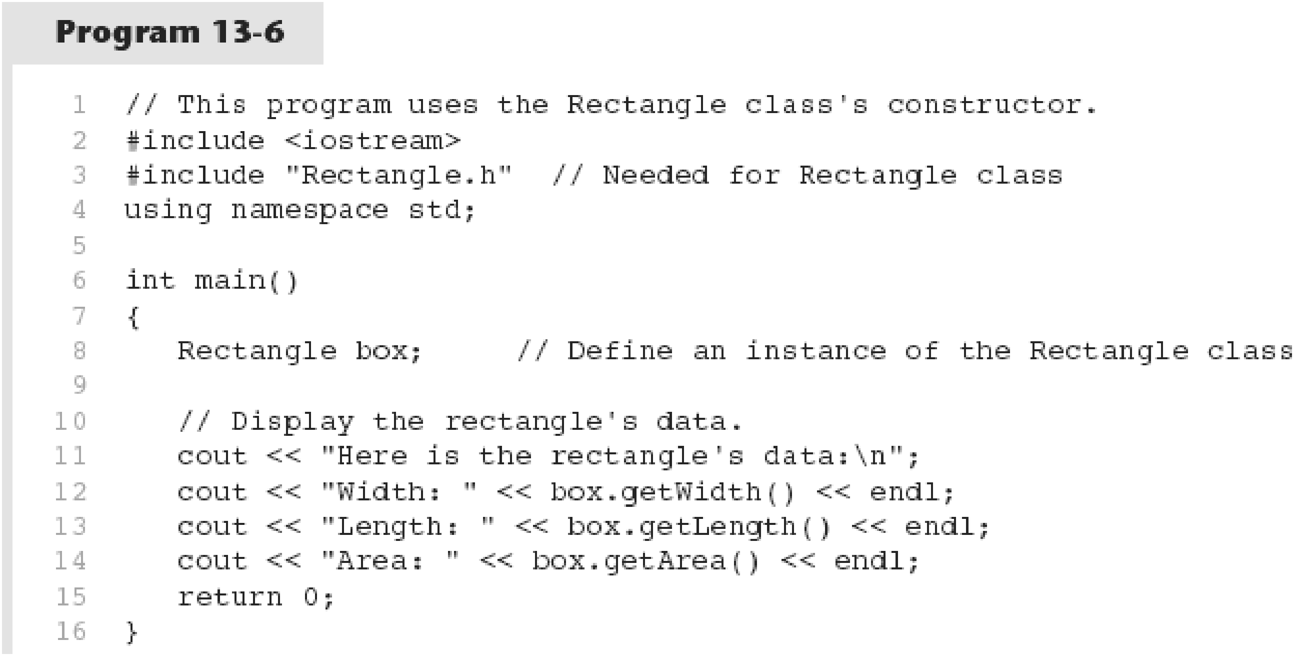
Contents of `Rectangle.ccp` Version3



```
//*************************************************************
// 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);
   }
}
```

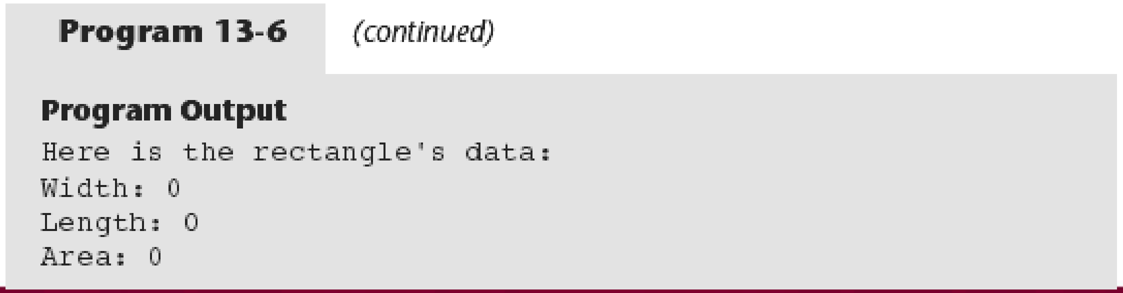
Example 1: Constructors (cont')

Contents of `Rectangle.ccp` Version3



Program 13-6

```
// 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 write a default constructor for you, one that does nothing.

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

```
Rectangle r;
```

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 (cont')

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

```
Rectangle r(10, 5);
```

More About Default Constructors

- If all of a constructor's parameters have default arguments, then it is a **default constructor**. For example:

```
Rectangle::Rectangle(double w=0.0,
 double len=0.0){
      width = w;
      length = len;
}
```

  - Creating an object and passing no arguments will cause this constructor to execute.

```
    Rectangle r;
```

Classes with No Default Constructor

- When all of a **class's constructors require arguments**, 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.

Destructors

Destructors

- Member function automatically called when an object is **destroyed**

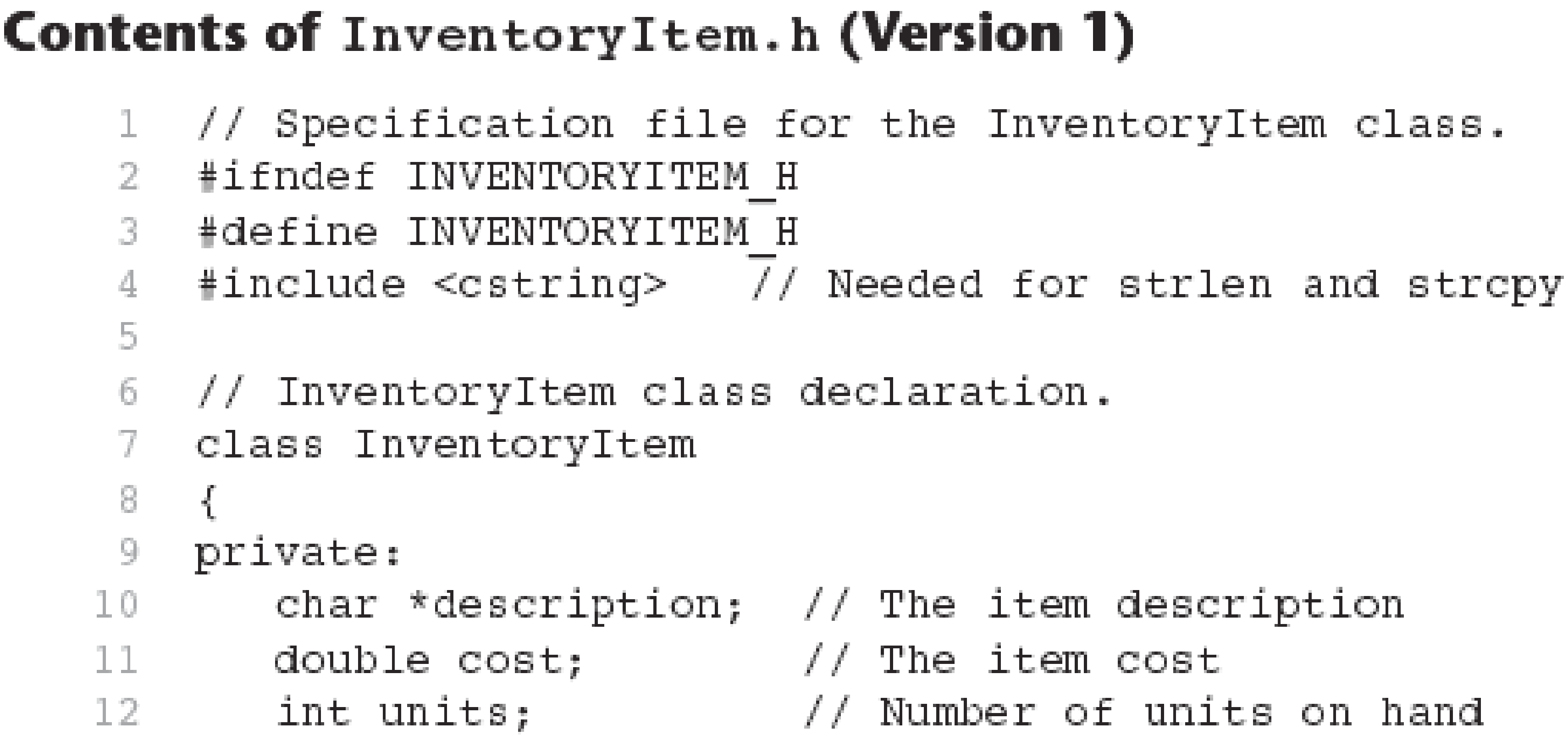
- Destructor name is **~classname**, *e.g.*, `~Rectangle`

- Has no return type; takes **no arguments**.

- **Only one destructor** per class, *i.e.*, it cannot be overloaded.

- If constructor allocates dynamic memory, destructor should release it.

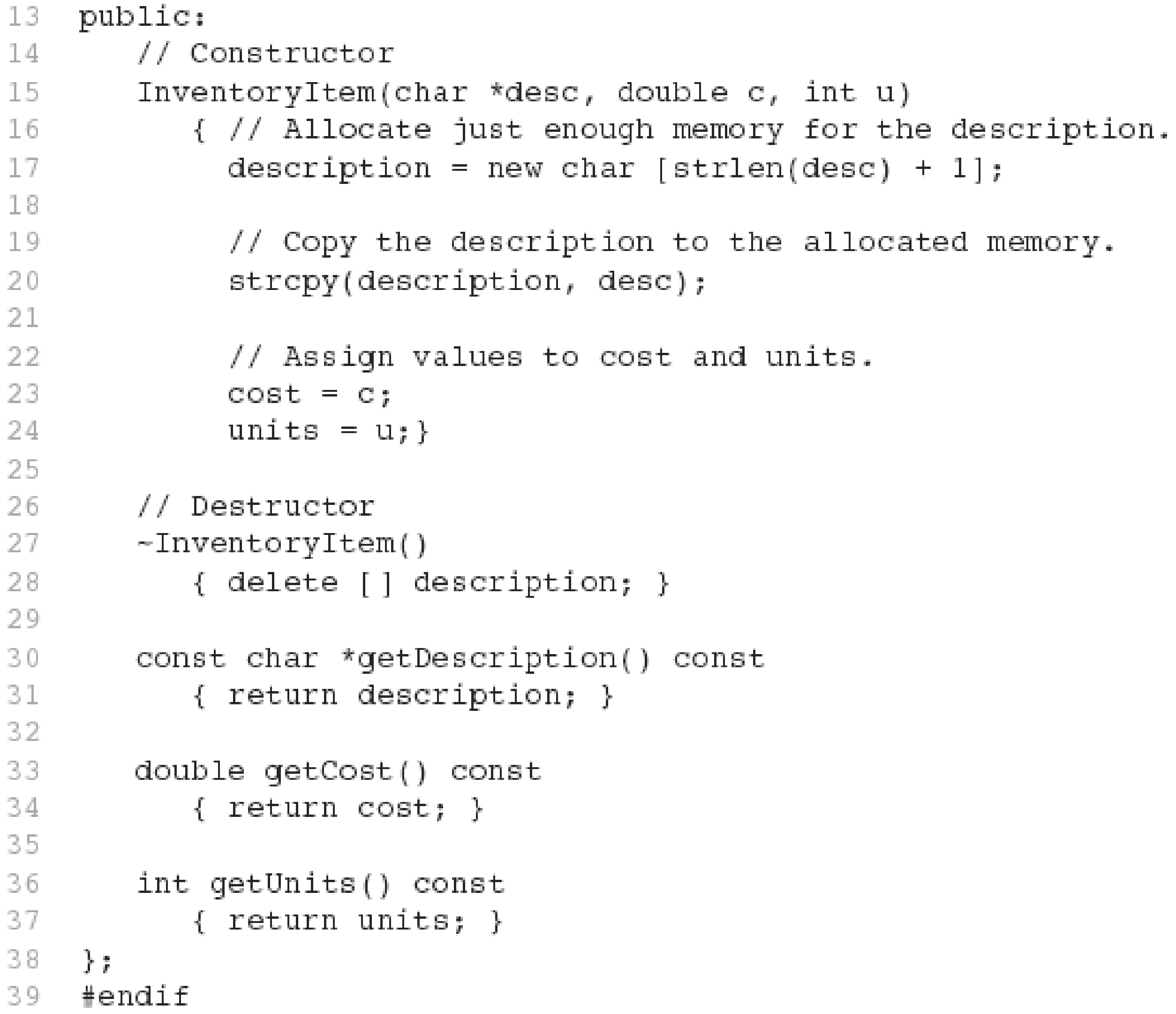
Example 2: Destructors



Contents of `InventoryItem.h` (Version 1)

```
// Specification file for the InventoryItem class.
#ifndef INVENTORYITEM_H
#define INVENTORYITEM_H
#include <cstring>   // Needed for strlen and strcpy

// InventoryItem class declaration.
class InventoryItem
{
private:
   char *description;  // The item description
   double cost;        // The item cost
   int units;          // Number of units on hand
```

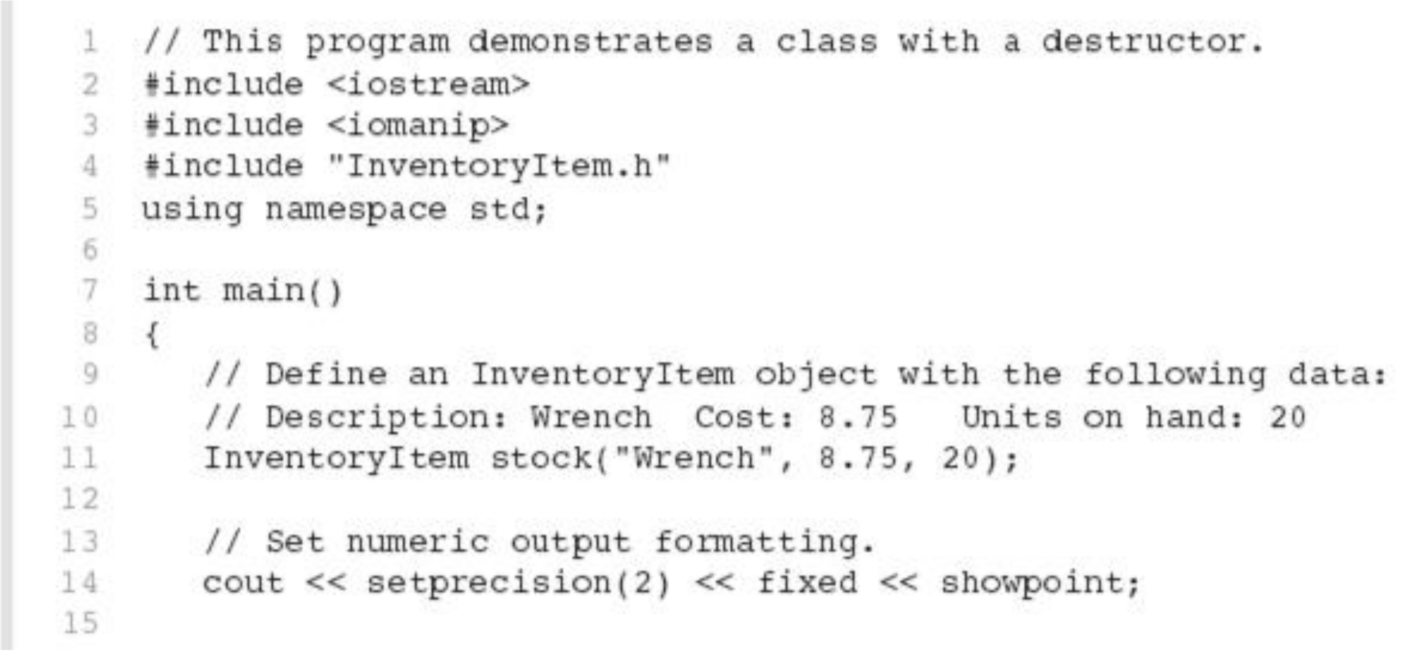
Example 2: Destructors



```cpp
public:
   // Constructor
   InventoryItem(char *desc, double c, int u)
      { // Allocate just enough memory for the description.
        description = new char [strlen(desc) + 1];

        // Copy the description to the allocated memory.
        strcpy(description, desc);

        // Assign values to cost and units.
        cost = c;
        units = u;}

   // Destructor
   ~InventoryItem()
      { delete [] description; }

   const char *getDescription() const
      { return description; }

   double getCost() const
      { return cost; }

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

Example 2: Destructors

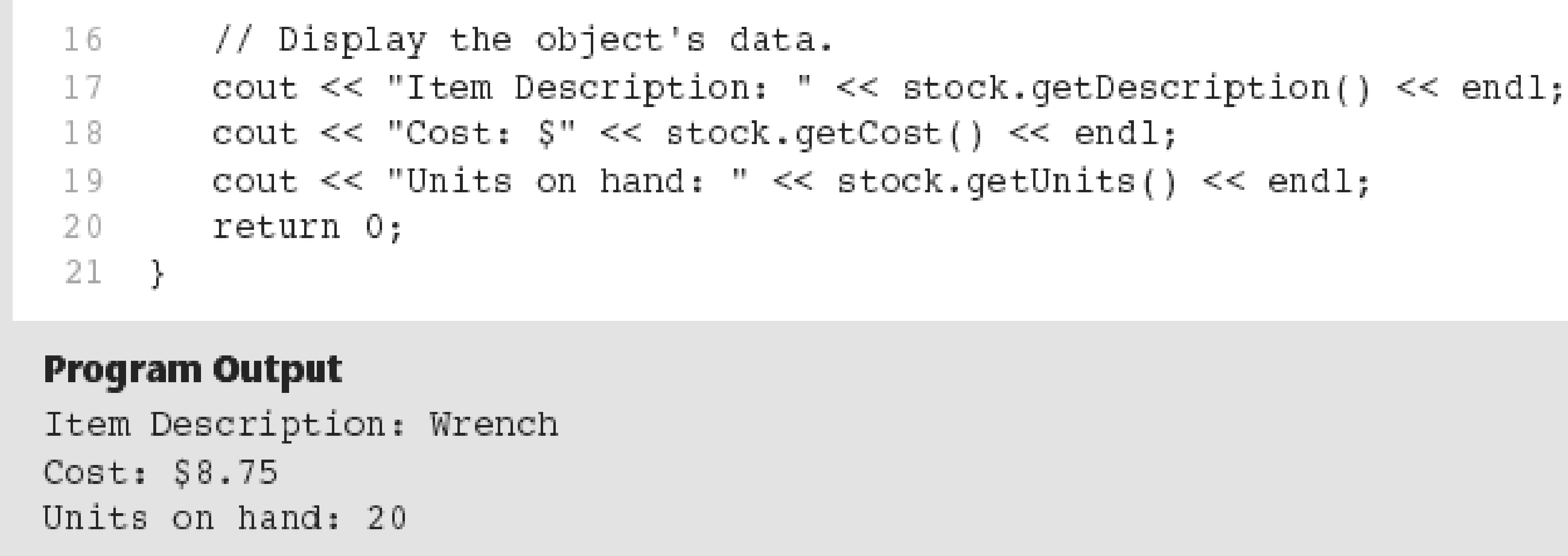
Contents of `InventoryItem.h` Version1 (cont’)



```
// This program demonstrates a class with a destructor.
#include <iostream>
#include <iomanip>
#include "InventoryItem.h"
using namespace std;

int main()
{
   // Define an InventoryItem object with the following data:
   // Description: Wrench  Cost: 8.75   Units on hand: 20
   InventoryItem stock("Wrench", 8.75, 20);

   // Set numeric output formatting.
   cout << setprecision(2) << fixed << showpoint;

```

Example 2: Destructors

Contents of `InventoryItem.h` Version1 (cont’)



```
    // Display the object's data.
    cout << "Item Description: " << stock.getDescription() << endl;
    cout << "Cost: $" << stock.getCost() << endl;
    cout << "Units on hand: " << stock.getUnits() << endl;
    return 0;
}
```

Program Output

```
Item Description: Wrench
Cost: $8.75
Units on hand: 20
```

Constructors, Destructors, and Dynamically Allocated Objects

- When an object is dynamically allocated with the new operator, its constructor executes:

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

- When the object is destroyed, its destructor executes:

```
delete r;
```

Overloading Constructors

Overloading Constructors

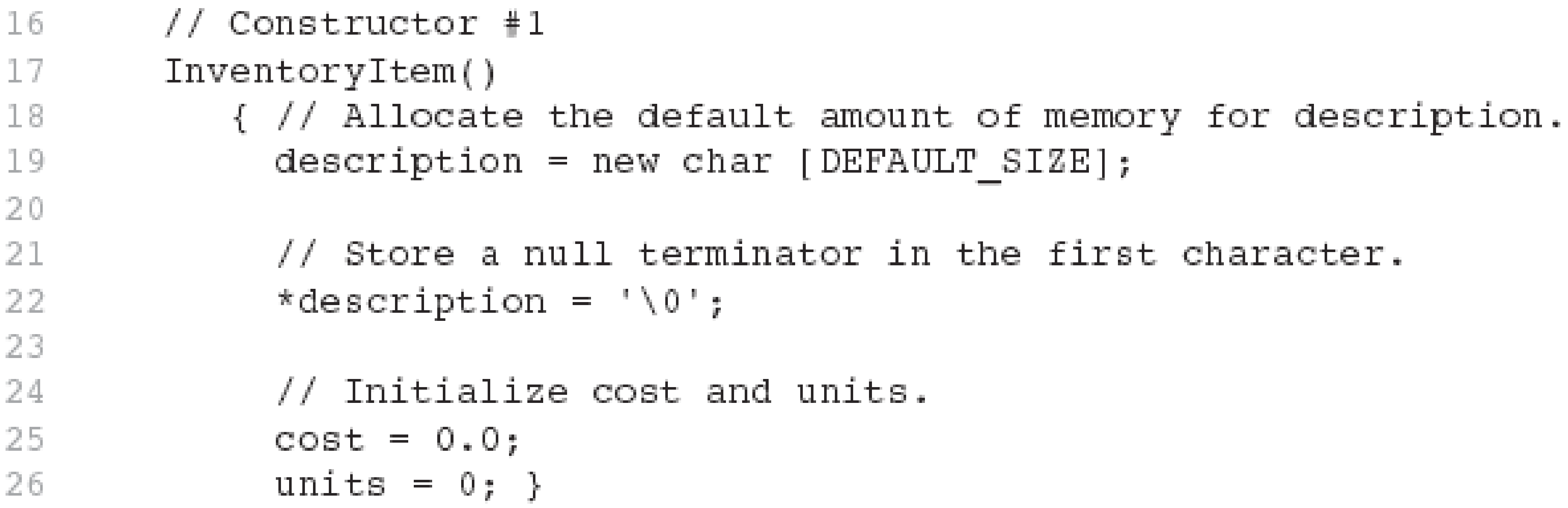
- A class can have more than one constructor.

- Overloaded constructors in a class must have different parameter lists:

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

Program 1

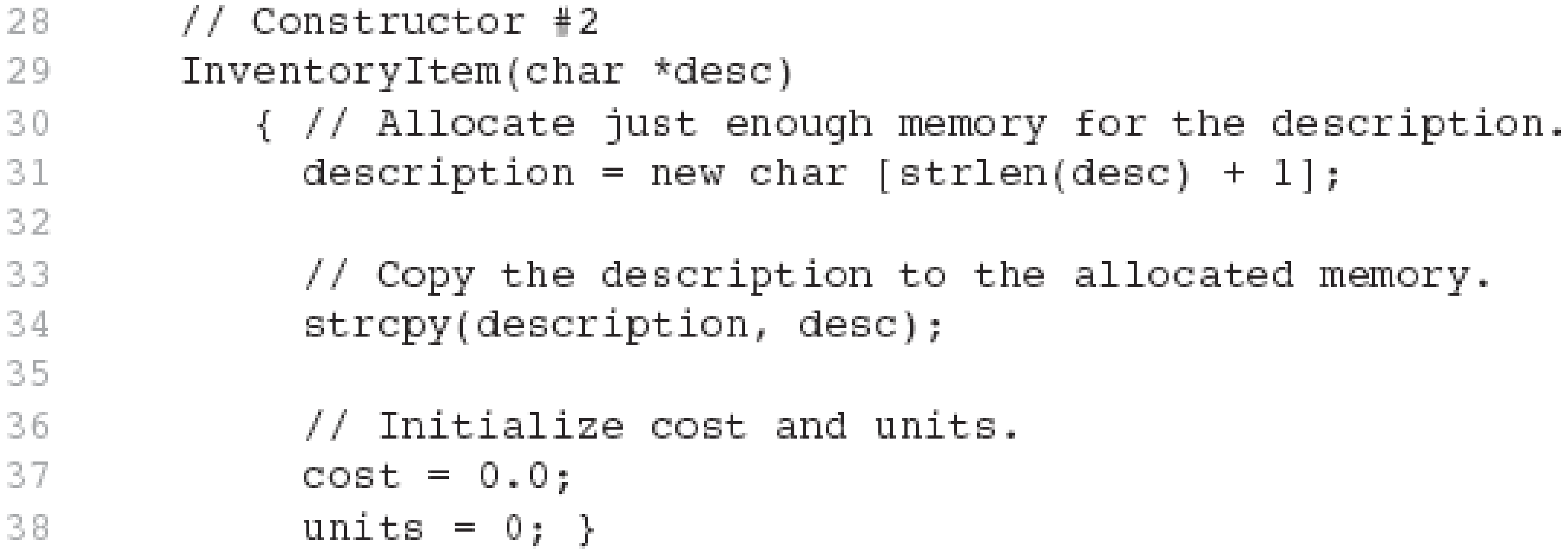
From Contents of `InventoryItem.h` Version2



```
// Constructor #1
InventoryItem()
   { // Allocate the default amount of memory for description.
     description = new char [DEFAULT_SIZE];

     // Store a null terminator in the first character.
     *description = '\0';

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

Program 1

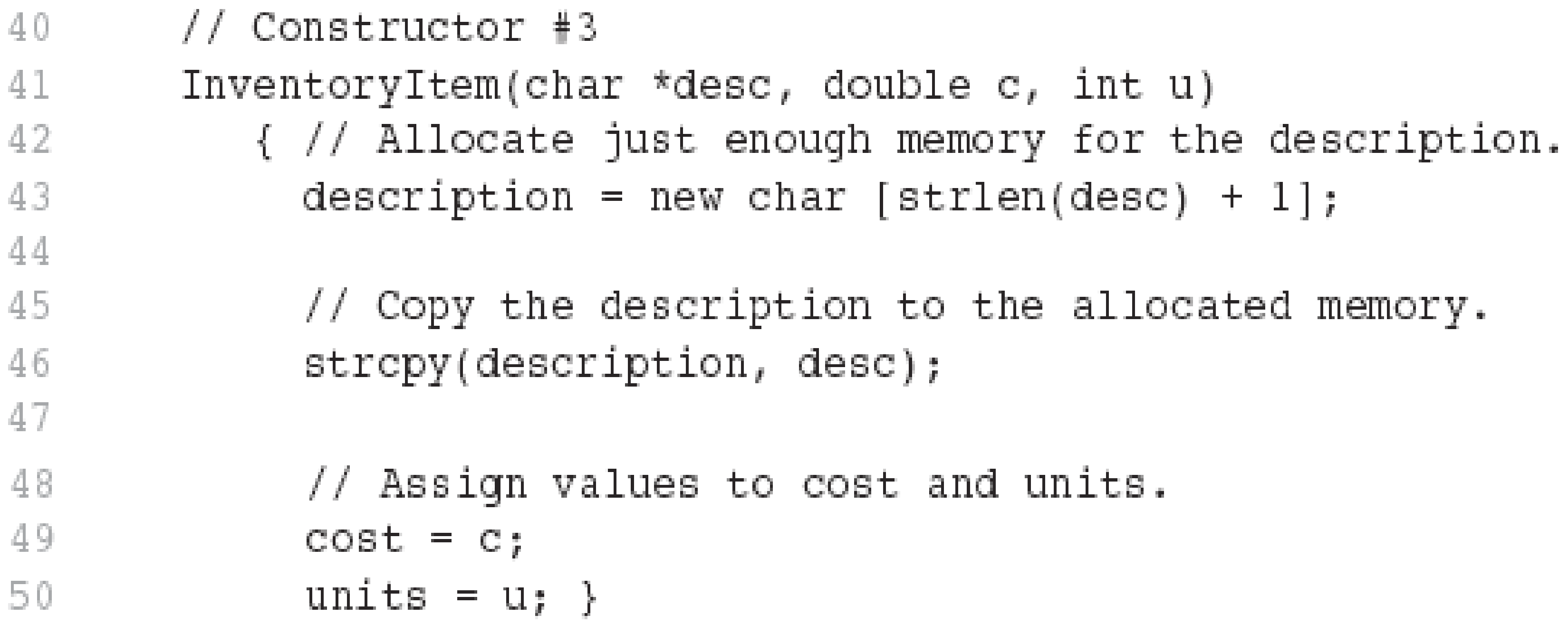
From Contents of `InventoryItem.h` Version2



```
    // Constructor #2
    InventoryItem(char *desc)
       { // Allocate just enough memory for the description.
         description = new char [strlen(desc) + 1];

         // Copy the description to the allocated memory.
         strcpy(description, desc);

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

Program 1

From Contents of `InventoryItem.h` Version2



```
// Constructor #3
InventoryItem(char *desc, double c, int u)
   { // Allocate just enough memory for the description.
     description = new char [strlen(desc) + 1];

     // Copy the description to the allocated memory.
     strcpy(description, desc);

     // Assign values to cost and units.
     cost = c;
     units = u; }
```

Only One Default Constructor and One Destructor

- **Do not provide more than one default** constructor for a class: one that takes no arguments and one that has default arguments for all parameters.

```
    Square();
    Square(int = 0);   // will not compile
```

- Since a destructor takes no arguments, there can only be one destructor for a class.

Member Function Overloading

- Non-constructor member functions can also be overloaded:

```
void setCost(double);
void setCost(char *);
```

- Must have unique parameter lists as for constructors.

Example 3: Member Function Overloading

```
#include <iostream>
using namespace std;
class Rectangle
{       private:   int height, width;
      public:
         Rectangle(int);
         Rectangle(int, int);
         int getSide()
         {return height;}
         int getArea(int);
         int getArea(int, int);
};
```

Example 3: Member Function Overloading (cont')

```
Rectangle::Rectangle(int x)
{   height=x;    width=x;}
Rectangle::Rectangle(int x, int y)
{    height=x;       width=y;}
int Rectangle::getArea(int x)
{    return(x*x);}
int Rectangle::getArea(int x, int y)
{    return (x*y);}
int main(){
    Rectangle c(5,6);
    Rectangle d(6);
    cout<<d.getArea(d.getSide())<<endl;
    cout<<c.getArea(5,6);
    return 0;
}
```

Constructor overloading

Function overloading

Using Private Member Functions

- A `private` member function can only be called by another member function.

- It is used for internal processing by the class, not for use outside of the class.

- See the `createDescription` function in **`InventoryItem.h`** (Version 3).

From Contents of InventoryItem.h Version2

```
class InventoryItem
{
private:
   char *description;   // The item description
   double cost;         // The item cost
   int units;           // Number of units on hand

void createDescription(int size, char *value)   {
    description = new char [size];
    strcpy(description, value); }

public:
   :
   :

 void setDescription(char *d)
  { strcpy(description, d); }
:
:
};
```

Copy Constructors

Copy Constructors

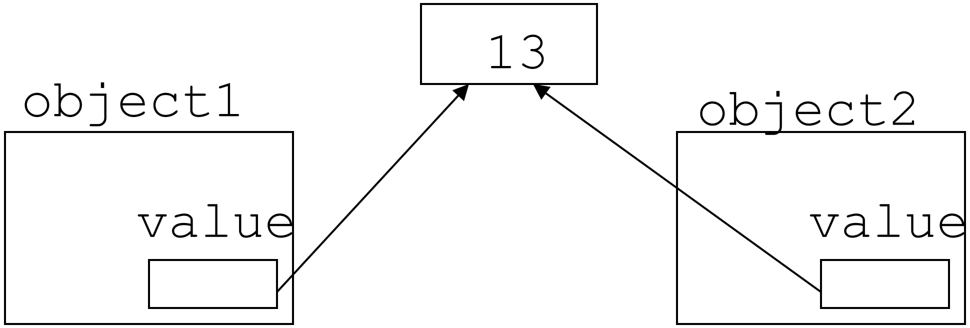
Problem: what if object contains a pointer?

```
class SomeClass
{ public:
     SomeClass(int val = 0)
        {value = new int; *value = val;}
     int getVal();
     void setVal(int);
    private:
     int *value;
}
```

Copy Constructors

- What we get using memberwise copy with objects containing dynamic memory:

```
SomeClass object1(5);
SomeClass object2 = object1;
object2.setVal(13);
cout << object1.getVal(); // also 13
```



Programmer-Defined Copy Constructor

- Allows us to solve problem with objects containing pointers:

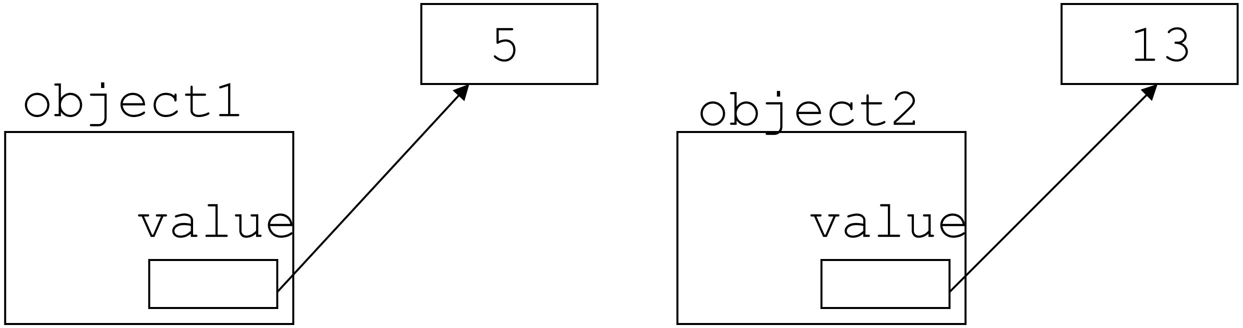
```
SomeClass::SomeClass(const SomeClass
&obj)
{
        value = new int;
        *value = *obj.value;
 }
```

- Copy constructor takes a reference parameter to an object of the class.

Programmer-Defined Copy Constructor

- Each object now points to separate dynamic memory:

```
SomeClass object1(5);
SomeClass object2 = object1;
object2.setVal(13);
cout << object1.getVal();
// still 5
```



Programmer-Defined Copy Constructor

- Since copy constructor has a reference to the object it is copying from,

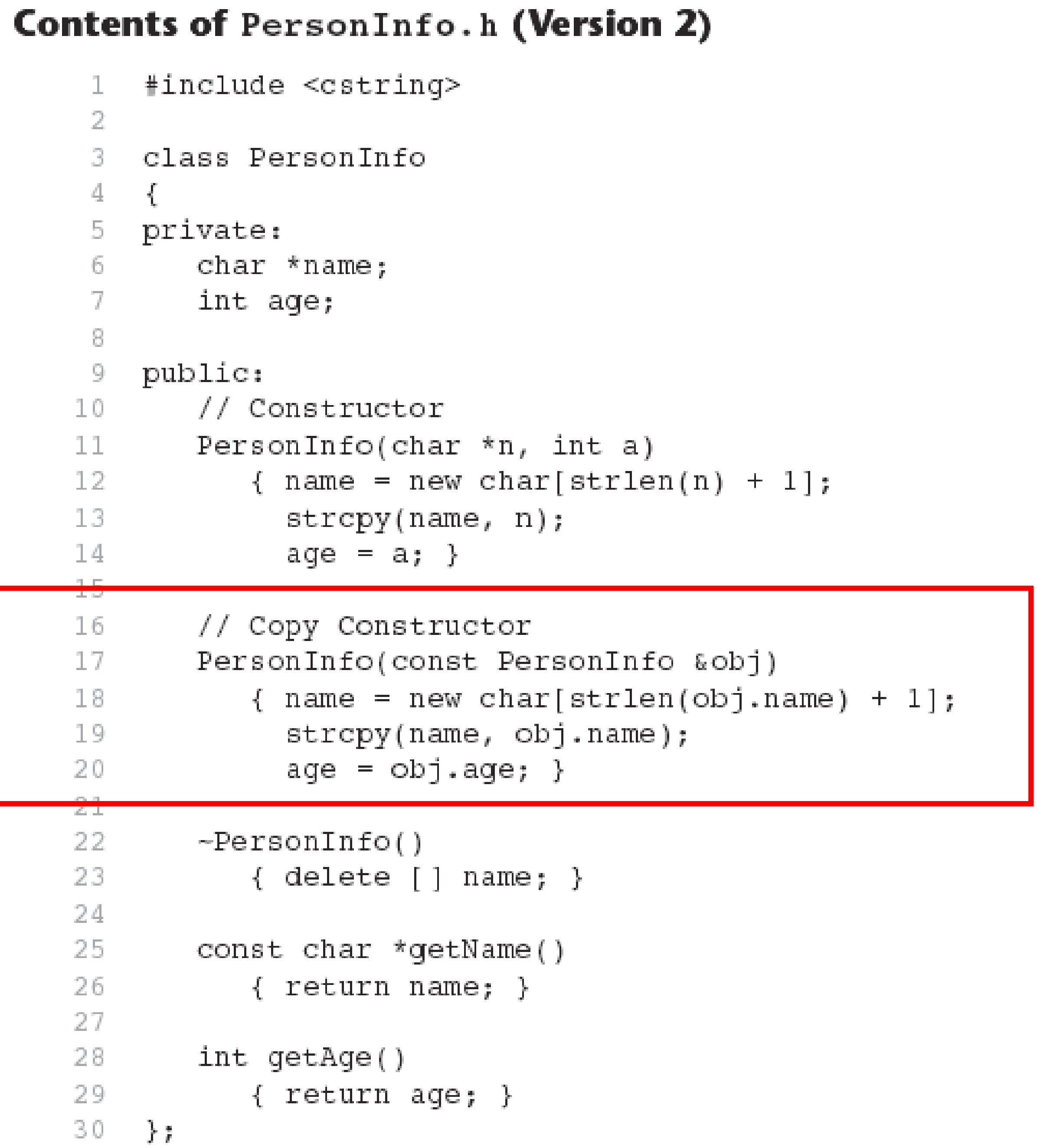
```
SomeClass::SomeClass(SomeClass &obj)
```

it can modify that object.

- To prevent this from happening, make the object parameter `const`:

```
SomeClass::SomeClass(const SomeClass &obj)
```

Example 5: Copy Constructor



Contents of PersonInfo.h (Version 2)

```
#include <cstring>

class PersonInfo
{
private:
   char *name;
   int age;

public:
   // Constructor
   PersonInfo(char *n, int a)
      { name = new char[strlen(n) + 1];
        strcpy(name, n);
        age = a; }

   // Copy Constructor
   PersonInfo(const PersonInfo &obj)
      { name = new char[strlen(obj.name) + 1];
        strcpy(name, obj.name);
        age = obj.age; }

   ~PersonInfo()
      { delete [] name; }

   const char *getName()
      { return name; }

   int getAge()
      { return age; }
};
```

In-Class Practice: Understanding Copy Constructors

- Using pair programming
- Create a class named Book that represents a book with the following properties:
  - title (string) - char *title
  - author (string) - char *author
  - pages (int) - int pages
- Implement the class which include:
  - A constructor that initializes the title, author, and pages.
  - A copy constructor that creates a copy of another Book object.
  - A method to display the book details.
- Test the copy constructor:
  - In the main function, create an instance of Book and then create another instance by copying the first instance using the copy constructor.
  - Print the details of both books to demonstrate that the copy constructor works correctly.
- Do all the code in one file, no need to separate specification and implementation.