

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)

```
1 // Specification file for the Rectangle class
2 // This version has a constructor.
3 #ifndef RECTANGLE_H
4 #define RECTANGLE_H
5
6 class Rectangle
7 {
8     private:
9         double width;
10        double length;
11    public:
12        Rectangle();           // Constructor
13        void setWidth(double);
14        void setLength(double);
15
16        double getWidth() const
17            { return width; }
18
19        double getLength() const
20            { return length; }
21
22        double getArea() const
23            { return width * length; }
24 };
25 #endif
```

**Constructor
in class definition**

Example 1: Constructors (cont')

Contents of Rectangle.cpp (Version 3)

```
1  // Implementation file for the Rectangle class.
2  // This version has a constructor.
3  #include "Rectangle.h"    // Needed for the Rectangle class
4  #include <iostream>       // Needed for cout
5  #include <cstdlib>        // Needed for the exit function
6  using namespace std;
7
8  /*******
9  // The constructor initializes width and length to 0.0.      *
10 /*******
11
12 Rectangle::Rectangle()
13 {
14     width = 0.0;
15     length = 0.0;
16 }
```

**Constructor
definition**

Example 1: Constructors (cont')

Contents of Rectangle.ccp Version3

```
17
18 //*****
19 // setWidth sets the value of the member variable width.    *
20 //*****
21
22 void Rectangle::setWidth(double w)
23 {
24     if (w >= 0)
25         width = w;
26     else
27     {
28         cout << "Invalid width\n";
29         exit(EXIT_FAILURE);
30     }
31 }
32
33 //*****
34 // setLength sets the value of the member variable length.  *
35 //*****
36
37 void Rectangle::setLength(double len)
38 {
39     if (len >= 0)
40         length = len;
41     else
42     {
43         cout << "Invalid length\n";
44         exit(EXIT_FAILURE);
45     }
46 }
```

Example 1: Constructors (cont')

Contents of Rectangle.ccp Version3

Program 13-6

```
1  // This program uses the Rectangle class's constructor.
2  #include <iostream>
3  #include "Rectangle.h"  // Needed for Rectangle class
4  using namespace std;
5
6  int main()
7  {
8      Rectangle box;      // Define an instance of the Rectangle class
9
10     // Display the rectangle's data.
11     cout << "Here is the rectangle's data:\n";
12     cout << "Width: " << box.getWidth() << endl;
13     cout << "Length: " << box.getLength() << endl;
14     cout << "Area: " << box.getArea() << endl;
15     return 0;
16 }
```

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)

```
1  // Specification file for the InventoryItem class.
2  #ifndef INVENTORYITEM_H
3  #define INVENTORYITEM_H
4  #include <cstring>    // Needed for strlen and strcpy
5
6  // InventoryItem class declaration.
7  class InventoryItem
8  {
9  private:
10     char *description;    // The item description
11     double cost;          // The item cost
12     int units;            // Number of units on hand
```

Example 2: Destructors

```
13 public:
14     // Constructor
15     InventoryItem(char *desc, double c, int u)
16     { // Allocate just enough memory for the description.
17         description = new char [strlen(desc) + 1];
18
19         // Copy the description to the allocated memory.
20         strcpy(description, desc);
21
22         // Assign values to cost and units.
23         cost = c;
24         units = u;}
25
26     // Destructor
27     ~InventoryItem()
28     { delete [] description; }
29
30     const char *getDescription() const
31     { return description; }
32
33     double getCost() const
34     { return cost; }
35
36     int getUnits() const
37     { return units; }
38 };
39 #endif
```

Example 2: Destructors

Contents of InventoryItem.h Version1 (cont')

```
1  // This program demonstrates a class with a destructor.
2  #include <iostream>
3  #include <iomanip>
4  #include "InventoryItem.h"
5  using namespace std;
6
7  int main()
8  {
9      // Define an InventoryItem object with the following data:
10     // Description: Wrench  Cost: 8.75  Units on hand: 20
11     InventoryItem stock("Wrench", 8.75, 20);
12
13     // Set numeric output formatting.
14     cout << setprecision(2) << fixed << showpoint;
15
```

Example 2: Destructors

Contents of InventoryItem.h Version1 (cont')

```
16      // Display the object's data.
17      cout << "Item Description: " << stock.getDescription() << endl;
18      cout << "Cost: $" << stock.getCost() << endl;
19      cout << "Units on hand: " << stock.getUnits() << endl;
20      return 0;
21  }
```

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

```
16      // Constructor #1
17      InventoryItem()
18      { // Allocate the default amount of memory for description.
19          description = new char [DEFAULT_SIZE];
20
21          // Store a null terminator in the first character.
22          *description = '\0';
23
24          // Initialize cost and units.
25          cost = 0.0;
26          units = 0; }
```

Program 1

From Contents of InventoryItem.h Version2

```
28      // Constructor #2
29      InventoryItem(char *desc)
30      { // Allocate just enough memory for the description.
31          description = new char [strlen(desc) + 1];
32
33          // Copy the description to the allocated memory.
34          strcpy(description, desc);
35
36          // Initialize cost and units.
37          cost = 0.0;
38          units = 0; }
```

Program 1

From Contents of InventoryItem.h Version2

```
40      // Constructor #3
41      InventoryItem(char *desc, double c, int u)
42      { // Allocate just enough memory for the description.
43          description = new char [strlen(desc) + 1];
44
45          // Copy the description to the allocated memory.
46          strcpy(description, desc);
47
48          // Assign values to cost and units.
49          cost = c;
50          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 2).

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

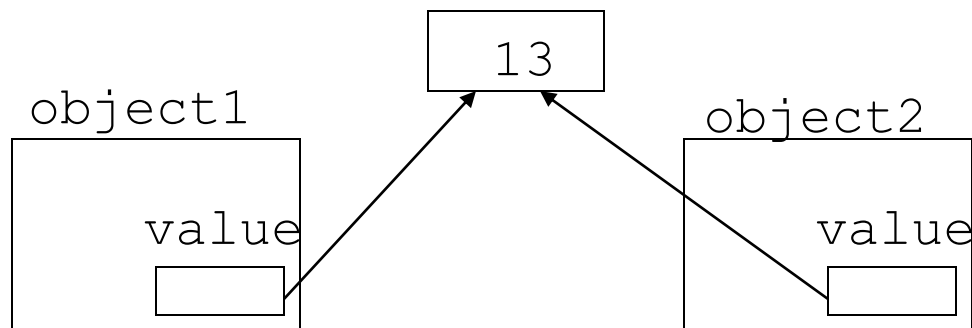
Problem: what if object contains a pointer?

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

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:

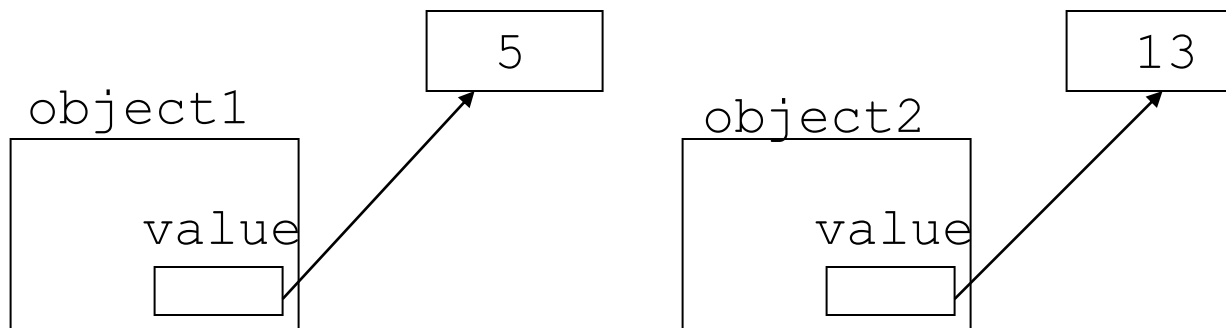
```
SomeClass::SomeClass(const SomeClass  
&obj)  
{  
    value = new int;  
    *value = *obj.value;  
}
```

✿ 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)

```
1  #include <cstring>
2
3  class PersonInfo
4  {
5  private:
6      char *name;
7      int age;
8
9  public:
10     // Constructor
11     PersonInfo(char *n, int a)
12     { name = new char[strlen(n) + 1];
13       strcpy(name, n);
14       age = a; }
15
16     // Copy Constructor
17     PersonInfo(const PersonInfo &obj)
18     { name = new char[strlen(obj.name) + 1];
19       strcpy(name, obj.name);
20       age = obj.age; }
21
22     ~PersonInfo()
23     { delete [] name; }
24
25     const char *getName()
26     { return name; }
27
28     int getAge()
29     { return age; }
30 };
```