جَامعة
المَنارة
MANARA UNIVERSITY

Introduction to Classes

Chapter 13

13.1 Procedural and Object-Oriented Programming

- Procedural programming is a method of writing software. It is a programming practice centered on the procedures, or actions that take place in a program.
- Object-Oriented programming is centered around the object. Objects are created form abstract data types that encapsulate data and functions together.

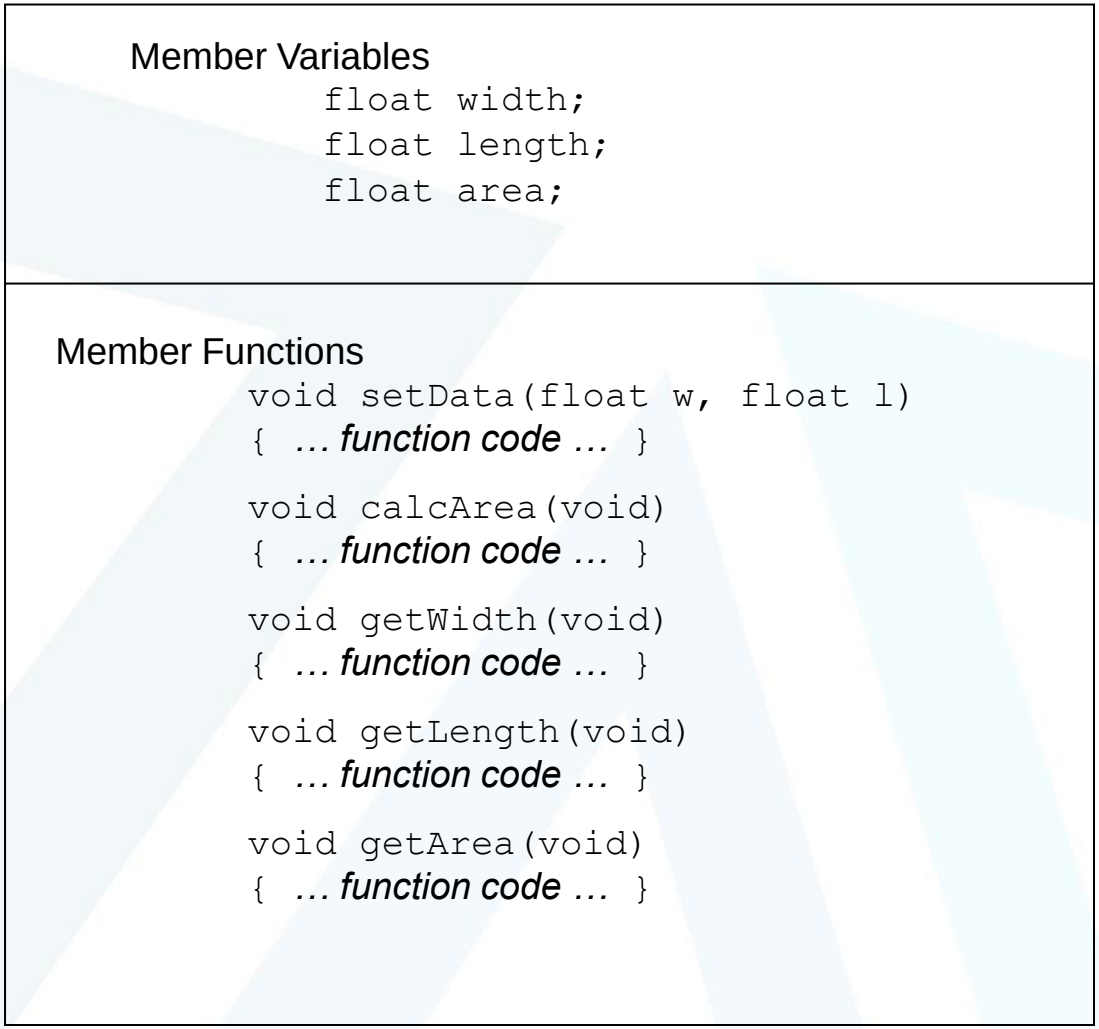
What's Wrong with Procedural Programming?

- Programs with excessive global data
- Complex and convoluted programs
- Programs that are difficult to modify and extend

What is Object-Oriented Programming?

- OOP is centered around the object, which packages together both the data and the functions that operate on the data.

Figure 13-1


Member Variables
float width;
float length;
float area;
Member Functions
void setData(float w, float l)
{ … function code … }
void calcArea(void)
{ … function code … }
void getWidth(void)
{ … function code … }
void getLength(void)
{ … function code … }
void getArea(void)
{ … function code … }

Terminology

- In OOP, an object's member variables are often called its *attributes* and its member functions are sometimes referred to as its *behaviors* or *methods*.

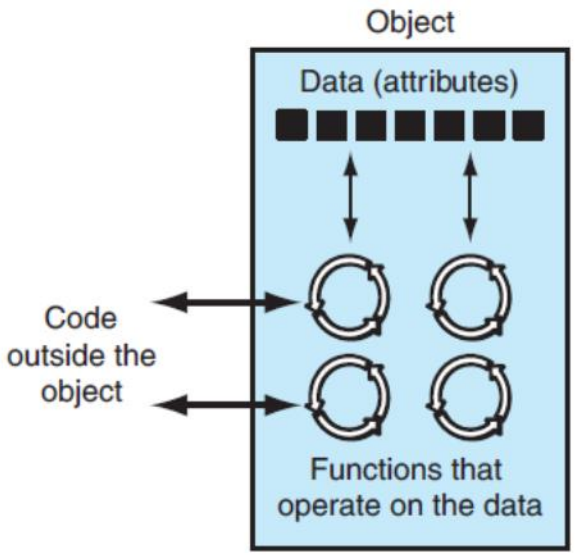
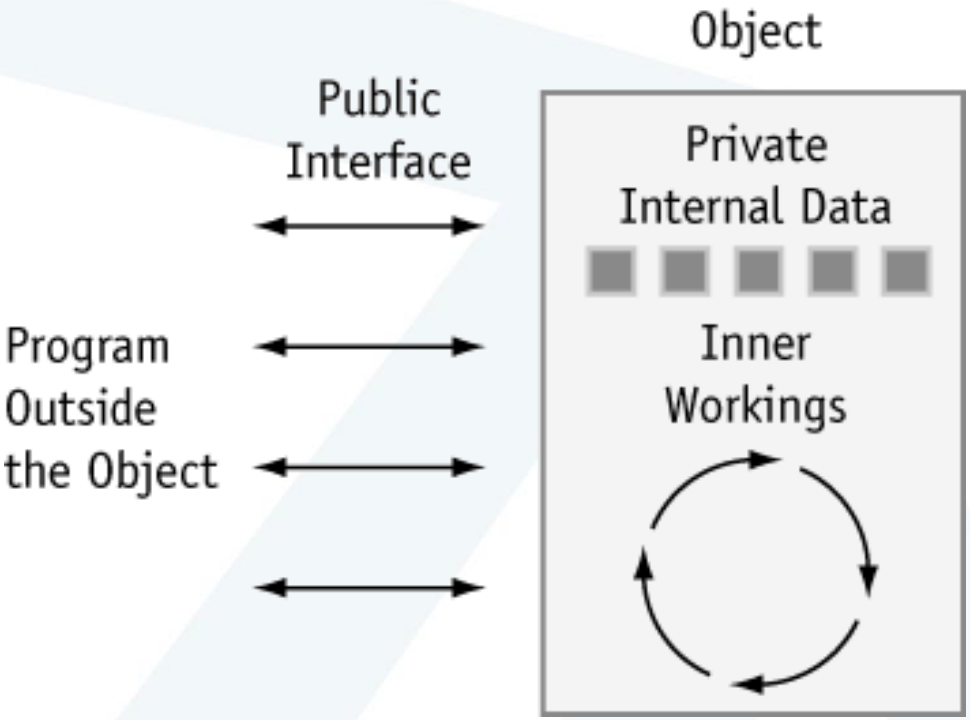


Figure 13-2


Object
Public
Interface
Private
Internal Data
Program
Outside
the Object
Inner
Workings

How are Objects Used?

- Although the use of objects is only limited by the programmer's imagination, they are commonly used to create data types that are either very specific or very general in purpose.

General Purpose Objects

- Creating data types that are improvements on C++'s built-in data types. For example, an array object could be created that works like a regular array, but additionally provides bounds-checking.
- Creating data types that are missing from C++. For instance, an object could be designed to process currencies or dates as if they were built-in data types.
- Creating objects that perform commonly needed tasks, such as input validation and screen output in a graphical user interface.

Application-Specific Objects

- Data types created for a specific application. For example, in an inventory program.

13.2 Introduction to the Class

- In C++, the class is the construct primarily used to create objects.

```
class class-name
{
  // declaration statements here
};
```

Example:

```
class Rectangle
{
      private:
         float width, length, area;
      public:
         void setData(float, float);
         void calcArea(void);
         float getWidth(void);
         float getLength(void);
         float getArea(void);
};
```

Access Specifiers

- The key words *private* and *public* are access specifiers.
- *private* means they can only be accessed by the member functions.
- *public* means they can be called from statements outside the class.
  - Note: the default access of a class is private, but it is still a good idea to use the private key word to explicitly declare private members. This clearly documents the access specification of the class.

13.3 Defining Member Functions

- Class member functions are defined similarly to regular functions.

```
void Rectangle::setData(float w, float l)
{
   width = w;
   length = l;
}
```

13.4 Defining an Instance of a Class

- Class objects must be defined after the class is declared.
- Defining a class object is called the instantiation of a class.
- Rectangle box; // box is an instance of Rectangle

Accessing an Object's Members

```
box.calcArea();
```

Pointers to Objects

```
Rectangle *boxPtr;
boxPtr = &box;
boxPtr->setData(15,12);
```

Program 13-1

```
// This program demonstrates a simple class.
#include <iostream.h>
// Rectangle class declaration.
class Rectangle
{
  private:
        float width;
        float length;
        float area;
  public:
        void setData(float, float);
        void calcArea(void);
        float getWidth(void);
        float getLength(void);
        float getArea(void);
};
```

Program continues

```
// setData copies the argument w to private member width and
// l to private member length.
void Rectangle::setData(float w, float l)
{
  width = w;
  length = l;
}

// calcArea multiplies the private members width and length.
// The result is stored in the private member area.
void Rectangle::calcArea(void)
{
  area = width * length;
}
```

Program continues

```
// getWidth returns the value in the private member width.
float Rectangle::getWidth(void)
{
  return width;
}
// getLength returns the value in the private member length.
float Rectangle::getLength(void)
{
  return length;
}
// getArea returns the value in the private member area.
float Rectangle::getArea(void)
{
  return area;
}
```

Program continues

```
void main(void)
{
   Rectangle box;
   float wide, long;
   cout << "This program will calculate the area of a\n";
   cout << "rectangle. What is the width? ";
   cin >> wide;
   cout << "What is the length? ";
   cin >> long;
   box.setData(wide, long);
   box.calcArea();
   cout << "Here is the rectangle's data:\n";
   cout << "width: " << box.getWidth() << endl;
   cout << "length: " << box.getLength() << endl;
   cout << "area: " << box.getArea() << endl;
}
```

Program Output

This program will calculate the area of a rectangle. What is the width? **10** [**Enter**]
What is the length? **5** [**Enter**]
Here is the rectangle's data:
width: 10
length: 5
area: 50

13.5 Why Have Private Members?

- In object-oriented programming, an object should protect its important data by making it private and providing a public interface to access that data.

13.6 Focus on Software Engineering: Some Design Considerations

- Usually class declarations are stored in their own header files. Member function definitions are stored in their own .CPP files.
- The #ifndef directive allows a program to be conditionally compiled. This prevents a header file from accidentally being included more than once.

Program 13-2

Contents of RECTANG.H

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
// Rectangle class declaration.
class Rectangle
{
   private:
          float width;
          float length;
          float area;
   public:
          void setData(float, float);
          void calcArea(void);
          float getWidth(void);
          float getLength(void);
          float getArea(void);
};
#endif
```

Program continues

Contents of RECTANG.CPP

```
#include "rectang.h"

// setData copies the argument w to private member width and
// l to private member length.
void Rectangle::setData(float w, float l)
{
  width = w;
  length = l;
}

// calcArea multiplies the private members width and length.
// The result is stored in the private member area.
void Rectangle::calcArea(void)
{
  area = width * length;
}
```

Program continues

```
// getWidth returns the value in the private member width.
float Rectangle::getWidth(void)
{
  return width;
}
// getLength returns the value in the private member length.
float Rectangle::getLength(void)
{
  return length;
}
// getArea returns the value in the private member area.
float Rectangle::getArea(void)
{
  return area;
}
```

Program continues

Contents of the main program, PR13-2.CPP

```
// This program demonstrates a simple class.
#include <iostream.h>
#include "rectang.h"  // contains Rectangle class declaration

// Don't forget to link this program with rectang.cpp!
void main(void)
{
   Rectangle box;
   float wide, long;
   cout << "This program will calculate the area of a\n";
   cout << "rectangle. What is the width? ";
   cin >> wide;
   cout << "What is the length? ";
   cin >> long;
```

Program continues

```
  box.setData(wide, long);
  box.calcArea();
  cout << "Here rectangle's data:\n";
  cout << "width: " << box.getWidth() << endl;
  cout << "length: " << box.getLength() << endl;
  cout << "area: " << box.getArea() << endl;
}
```

Performing I/O in a Class Object

- Notice that the Rectangle example has no cin or cout.
- This is so anyone who writes a program that uses the Rectangle class will not be “locked into” the way the class performs input or output.
- Unless a class is specifically designed to perform I/O, operations like user input and output are best left to the person designing the application.

Table 13-1

rectang.h	Contains the class definition of Rectangle. Is included by rectang.cpp and pr13-2.cpp
rectang.cpp	Contains Rectangle's member function definitions. Is compiled to an object file such as rectang.obj.
pr13-2.cpp	Contains function main. It is compiled to an object file, such as pr13-2.obj, which is linked with rectang.cpp's object file to form an executable file.
	rectang.cpp is compiled to rectang.obj pr13-2.cpp is compiled to pr13-2.obj pr13-2.obj and rectang.obj are linked to make pr13-2.exe

13.7 Focus on Software Engineering: Using Private Member Functions

- A private member function may only be called from a function that is a member of the same object.

Program 13-3

```
#include <iostream.h>
#include "rectang2.h"  // contains Rectangle class declaration
// Don't forget to link this program with rectang2.cpp!

void main(void)
{
   Rectangle box;
   float wide, long;
   cout << "This program will calculate the area of a\n";
   cout << "rectangle. What is the width? ";
   cin >> wide;
   cout << "What is the length? ";
   cin >> long;
   box.setData(wide, long);
   cout << "Here rectangle's data:\n";
   cout << "width: " << box.getWidth() << endl;
   cout << "length: " << box.getLength() << endl;
   cout << "area: " << box.getArea() << endl;
}
```

Program Output

This program will calculate the area of a rectangle. What is the width? **10** [**Enter**]
What is the length? **5** [**Enter**]
Here rectangle's data:
width: 10
length: 5
area: 50

13.8 Inline Member Functions

- When the body of a member function is defined inside a class declaration, it is declared inline.

Program 13-4

Contents of RECTANG3.H

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

// Rectangle class declaration.
class Rectangle
{
  private:
        float width;
        float length;
        float area;
        void calcArea(void) { area = width * length; }
```

Program continues

```
  public:
        void setData(float, float);   // Prototype
        float getWidth(void) { return width; }
        float getLength(void) { return length; }
        float getArea(void) { return area; }
};
#endif
```

Contents of rectang3.cpp

```
#include "rectang3.h"
// setData copies the argument w to private member width and
// l to private member length.
void Rectangle::setData(float w, float l)
{
  width = w;
  length = l;
  calcArea();
}
```

Program continues

Contents of the main program, pr13-4.cpp

```
#include <iostream.h>
#include "rectang3.h"  // contains Rectangle class declaration

// Don't forget to link this program with rectang3.cpp!

void main(void)
{
   Rectangle box;
   float wide, long;

   cout << "This program will calculate the area of a\n";
   cout << "rectangle. What is the width? ";
   cin >> wide;
   cout << "What is the length? ";
   cin >> long;
```

Program continues

```
  box.setData(wide, long);
  cout << "Here rectangle's data:\n";
  cout << "width: " << box.getWidth() << endl;
  cout << "length: " << box.getLength() << endl;
  cout << "area: " << box.getArea() << endl;
}
```

Program Output

This program will calculate the area of a rectangle. What is the width? **10** [**Enter**]
What is the length? **5** [**Enter**]
Here rectangle's data:
width: 10
length: 5
area: 50

13.9 Constructors

- A constructor is a member function that is automatically called when a class object is created.
- Constructors have the same name as the class.
- Constructors must be declared publicly.
- Constructors have no return type.

Program 13-5

```
// This program demonstrates a constructor.
#include <iostream.h>
class Demo
{
public:
  Demo(void);                                   // Constructor
};

Demo::Demo(void)
{
  cout << "Welcome to the constructor!\n";
}
```

Program continues

```
void main(void)
{
  Demo demoObj;   // Declare a Demo object;
  cout << "This program demonstrates an object\n";
  cout << "with a constructor.\n";
}
```

Program Output

```
Welcome to the constructor.
This program demonstrates an object
with a constructor.
```

Program 13-6

```
// This program demonstrates a constructor.
#include <iostream.h>

class Demo
{
public:
  Demo(void); // Constructor
};


Demo::Demo(void)
{
  cout << "Welcome to the constructor!\n";
}
```

Program continues

```
void main(void)
{
  cout << "This is displayed before the object\n";
  cout << "is declared.\n\n";
  Demo demoObj;
  cout << "\nThis is displayed after the object\n";
  cout << "is declared.\n";
}
```

Program Output

This is displayed before the object is declared.

Welcome to the constructor.

This is displayed after the object is declared.

Constructor Arguments

- When a constructor does not have to accept arguments, it is called an object's *default constructor*. Like regular functions, constructors may accept arguments, have default arguments, be declared inline, and be overloaded.

Program 13-7

```
// This program demonstrates a class with a constructor
#include <iostream.h>
#include <string.h>

class InvItem
{
   private:
          char *desc;
          int units;

   public:
          InvItem(void) { desc = new char[51]; }
          void setInfo(char *dscr, int un) { strcpy(desc, dscr);
                                    units = un;}
          char *getDesc(void) { return desc; }
          int getUnits(void) { return units; }
};
```

Program continues

```
void main(void)
{
  InvItem stock;
  stock.setInfo("Wrench", 20);
  cout << "Item Description: " << stock.getDesc() << endl;
  cout << "Units on hand: " << stock.getUnits() << endl;
}
```

Program Output

Item Description: Wrench

Units on hand: 20

13.10 Destructors

- A destructor is a member function that is automatically called when an object is destroyed.
  - Destructors have the same name as the class, preceded by a tilde character (~)
  - In the same way that a constructor is called then the object is created, the destructor is automatically called when the object is destroyed.
  - In the same way that a constructor sets things up when an object is created, a destructor performs shutdown procedures when an object is destroyed.

Program 13-8

```
// This program demonstrates a destructor.

#include <iostream.h>

class Demo
{
public:
   Demo(void);     // Constructor
   ~Demo(void);   // Destructor
};

Demo::Demo(void)
{
   cout << "Welcome to the constructor!\n";
}
```

Program continues

```
Demo::~Demo(void)
{
  cout << "The destructor is now running.\n";
}

void main(void)
{
  Demo demoObj;   // Declare a Demo object;
  cout << "This program demonstrates an object\n";
  cout << "with a constructor and destructor.\n";
}
```

Program Output

```
Welcome to the constructor!
This program demonstrates an object
with a constructor and destructor.
The destructor is now running.
```

Program 13-9

```
#include <iostream.h>
#include <string.h>
class InvItem
{
  private:
        char *desc;
        int units;
  public:
        InvItem(void) { desc = new char[51]; }
        ~InvItem(void) { delete desc; }
        void setInfo(char *dscr, int un) { strcpy(desc, dscr);
                                  units = un;}
        char *getDesc(void) { return desc; }
        int getUnits(void) { return units; }
};
```

Program continues

```
void main(void)
{
  InvItem stock;
  stock.setInfo("Wrench", 20);
  cout << "Item Description: " << stock.getDesc() << endl;
  cout << "Units on hand: " << stock.getUnits() << endl;
}
```

Program Output

Item Description: Wrench

Units on hand: 20

13.11 Constructors that Accept Arguments

- Information can be passed as arguments to an object's constructor.

Program 13-10

Contents of sale.h

```
#ifndef SALE_H
#define SALE_H

// Sale class declaration
class Sale
{
private:
   float taxRate;
   float total;
public:
   Sale(float rate) { taxRate = rate; }
   void calcSale(float cost)
            { total = cost + (cost * taxRate) };
   float getTotal(void) { return total; }
};
#endif
```

Program continues

Contents of main program, pr13-10.cpp

```
#include <iostream.h>
#include "sale.h"

void main(void)
{
  Sale cashier(0.06);  // 6% sales tax rate
  float amnt;
  cout.precision(2);
  cout.setf(ios::fixed | ios::showpoint);
  cout << "Enter the amount of the sale: ";
  cin >> amnt;
  cashier.calcSale(amnt);
  cout << "The total of the sale is $";
  cout << cashier.getTotal << endl;
}
```

Program Output

Enter the amount of the sale: 125.00

The total of the sale is $132.50

Program 13-11

Contents of sale2.h

```
#ifndef SALE2_H
#define SALE2_H

// Sale class declaration
class Sale
{
private:
   float taxRate;
   float total;
public:
   Sale(float rate = 0.05) { taxRate = rate; }
   void calcSale(float cost)
          { total = cost + (cost * taxRate) };
   float getTotal (void) { return total; }
};
#endif
```

Program continues

Contents of main program, pr13-11.cpp

```
#include <iostream.h>
#include "sale2.h"

void main(void)
{
  Sale cashier1;         // Use default sales tax rate
  Sale cashier2 (0.06);  // Use 6% sales tax rate
  float amnt;
  cout.precision(2);
  cout.set(ios::fixed | ios::showpoint);
  cout << "Enter the amount of the sale: ";
  cin >> amnt;
  cashier1.calcSale(amnt);
  cashier2.calcSale(amnt);
```

Program continues

```
  cout << "With a 0.05 sales tax rate, the total\n";
  cout << "of the sale is $";
  cout << cashier1.getTotal() << endl;
  cout << "With a 0.06 sales tax rate, the total\n";
  cout << "of the sale is $";
  cout << cashier2.getTotal() << endl;
}
```

Program Output

Enter the amount of the sale: 125.00
With a 0.05 sales tax rate, the total
of the sale is $131.25
With a 0.06 sales tax rate, the total
of the sale is $132.50

13.12 Focus on Software Engineering: Input Validation Objects

- This section shows how classes may be designed to validate user input.

Program 13-12

```
// This program demonstrates the CharRange class.
#include <iostream.h>
#include "chrange.h" // Remember to compile & link chrange.cpp

void main(void)
{
  // Create an object to check for characters
  // in the range J - N.
  CharRange input('J', 'N');

  cout << "Enter any of the characters J, K, l, M, or N.\n";
  cout << "Entering N will stop this program.\n";
  while (input.getChar() != 'N');
}
```

Program Output with Example Input

Enter any of the characters J, K, l, M, or N

Entering N will stop this program.

j

k

q

n [Enter]

13.13 Overloaded Constructors

- More than one constructor may be defined for a class.

Program 13-13

Contents of invitem2.h

```
#ifndef INVITEM2_H
#define INVITEM2_H
#include <string.h> // Needed for strcpy function call.

// InvItem class declaration
class InvItem
{
private:
  char *desc;
  int units;
public:
  InvItem(int size = 51) { desc = new char[size]; }
  InvItem(char *d) { desc = new char[strlen(d)+1];
                     strcpy(desc, d); }
```

Program continues

```
  ~InvItem(void) { delete[] desc; }
  void setInfo(char *d, int u) { strcpy(desc, d); units = u;}
  void setUnits (int u) { units = u; }
  char *getDesc(void) { return desc; }
  int getUnits(void) { return units; }
};
#endif
```

Contents of main program, pr13-13.cpp

```
// This program demonstrates a class with overloaded constructors
#include <iostream.h>
#include "invitem2.h"

void main(void)
{
```

Program continues

```
  InvItem item1("Wrench");
  InvItem item2;

  item1.setUnits(15);
  item2.setInfo("Pliers", 25);
  cout << "The following items are in inventory:\n";
  cout << "Description: " << item1.getDesc() << "\t\t";
  cout << "Units on Hand: " << item1.getUnits() << endl;
  cout << "Description: " << item2.getDesc() << "\t\t";
  cout << "Units on Hand: " << item2.getUnits() << endl;
}
```

Program Output

The following items are in inventory:

Description: Wrench Units on Hand: 15

Description: Pliers Units on Hand 25

13.14 Only One Default Constructor and one Destructor

- A class may only have one default constructor and one destructor.

13.15 Arrays of Objects

- You may declare and work with arrays of class objects.

InvItem inventory[40];

Program 13-14

Contents of invitem3.h

```
#ifndef INVITEM3_H
#define INVITEM3_H
#include <string.h>   // Needed for strcpy function call.

// InvItem class declaration
class InvItem
{
private:
  char *desc;
  int units;
public:
  InvItem(int size = 51) { desc = new char[size]; }
  InvItem(char *d) { desc = new[strlen(d)+1];
                     strcpy(desc, d); }
```

Program continues

```
  InvItem(char *d, int u) { desc = new[strlen(d)+1];
                            strcpy(desc, d);
                            units = u; }
  ~InvItem(void) { delete [] desc; }
  void setInfo(char * dscr, int u) { strcpy(desc, dscr); units = un;}
  void setUnits (int u) { units = u; }
  char *getDesc(void) { return desc; }
  int getUnits(void) { return units; }
};
#endif
```

Contents of main program, pr13-14.cpp

```
// This program demonstrates an array of objects.
#include <iostream.h>
#include <iomanip.h>
#include "invitem3.h"
```

Program continues

```
void main(void)
{
  InvItem Inventory[5] = { InvItem("Adjustable Wrench", 10),
                           InvItem("Screwdriver", 20), InvItem("Pliers", 35),
                           InvItem("Ratchet", 10), InvItem("Socket Wrench", 7)
                           };
  cout << "Inventory Item\t\tUnits On Hand\n";
  cout << "------------------------------\n";
  for (int Index = 0; Index < 5; Index++)
  {
        cout << setw(17) << Inventory[Index].GetDesc();
        cout << setw(12) << Inventory[Index].GetUnits()  << endl;
  }
}
```

Program Output

```
Inventory Item    Units On Hand
-----------------------------------
Adjustable Wrench          10
      Screwdriver          20
           Pliers          35
          Ratchet          10
    Socket Wrench           7
```