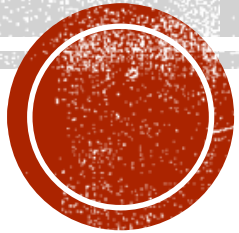
DATA STRUCTURES & ALGORITHMS

Lecture 01
Fahad Zafar



STRUCTURED DATA

- Abstract data types (ADTs) are data types created by the programmer. ADTs have their own range (or domain) of data and their own set of operations that may be performed on them.

ABSTRACTION

- An abstraction is a general model of something.

DATA TYPES

- C++ has several primitive data types:

Table 11-1

bool	int	unsigned long int
char	long int	float
unsigned char	unsigned short int	double
short int	unsigned int	long double

ABSTRACT DATA TYPES

- A data type created by the programmer
  - The programmer decides what values are acceptable for the data type
  - The programmer decides what operations may be performed on the data type

11.2 FOCUS ON SOFTWARE ENGINEERING: COMBINING DATA INTO STRUCTURES

- C++ allows you to group several variables together into a single item known as a structure.

TABLE 11-2

Variable Declaration	Information Held
int EmpNumber;	Employee Number
char Name[25];	Employee’s Name
float Hours;	Hours Worked
float PayRate;	Hourly Pay Rate
float GrossPay;	Gross Pay

TABLE 11-2 AS A STRUCTURE:

```
struct PayRoll
{
   int EmpNumber;
   char Name[25];
   float Hours;
   float PayRate;
   float GrossPay;
};
```

FIGURE 11-1

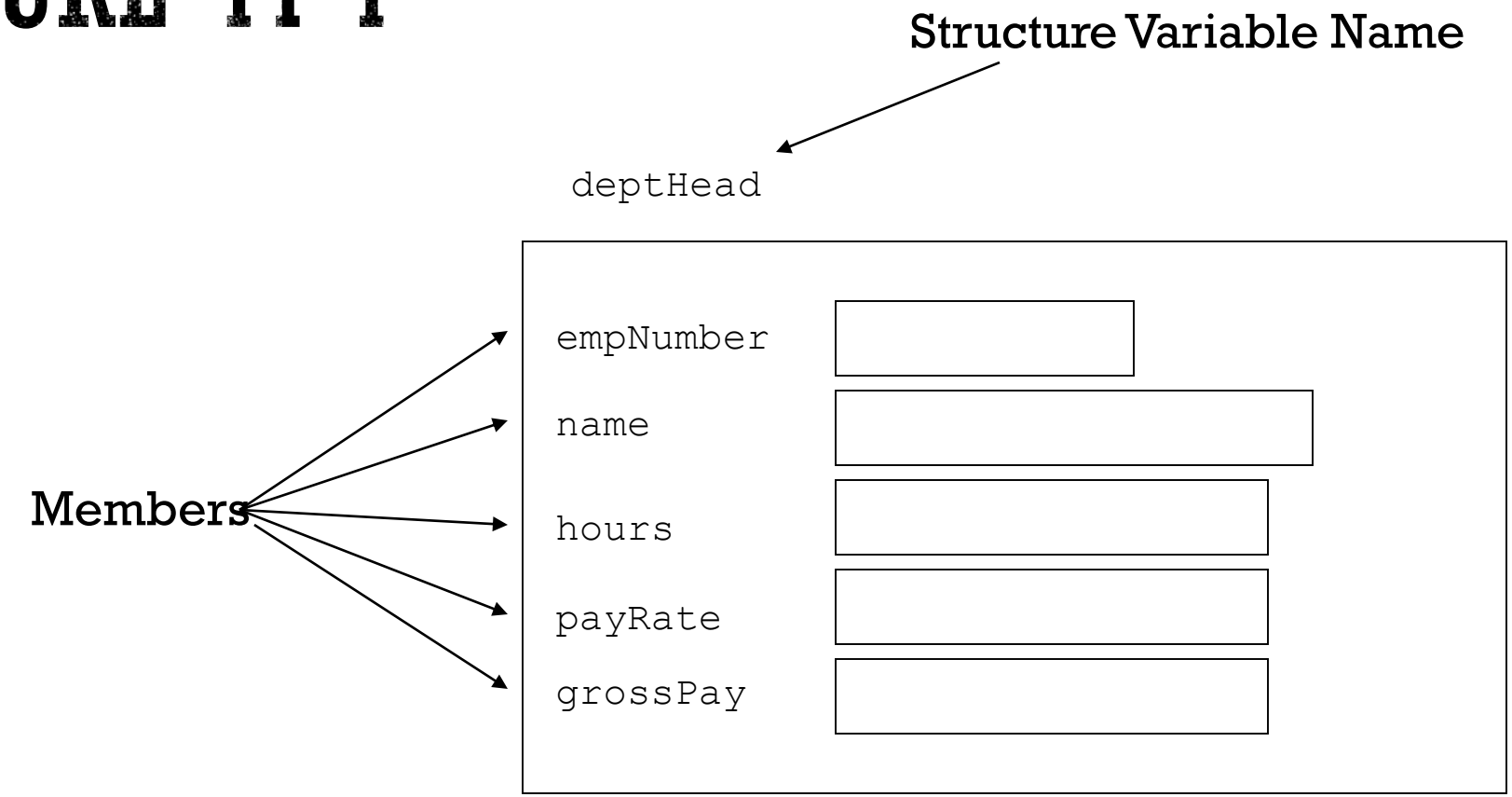
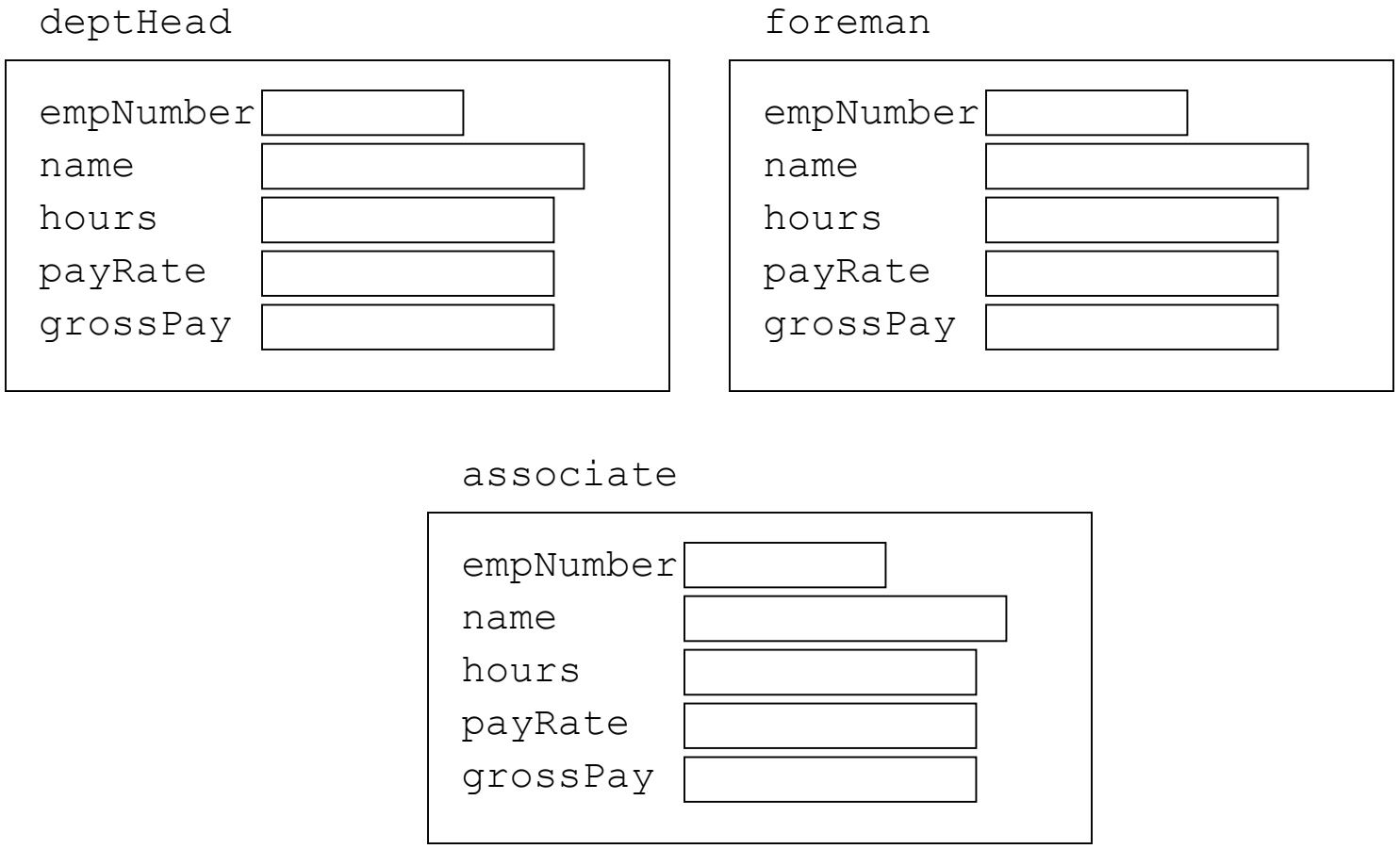

Structure Variable Name
deptHead
Members
empNumber
name
hours
payRate
grossPay

FIGURE 11-2


deptHead
empNumber
name
hours
payRate
grossPay
foreman
empNumber
name
hours
payRate
grossPay
associate
empNumber
name
hours
payRate
grossPay

TWO STEPS TO IMPLEMENTING STRUCTURES:

- Create the structure declaration. This establishes the tag (or name) of the structure and a list of items that are members.
- Declare variables (or instances) of the structure and use them in the program to hold data.

11.3 ACCESSING STRUCTURE MEMBERS

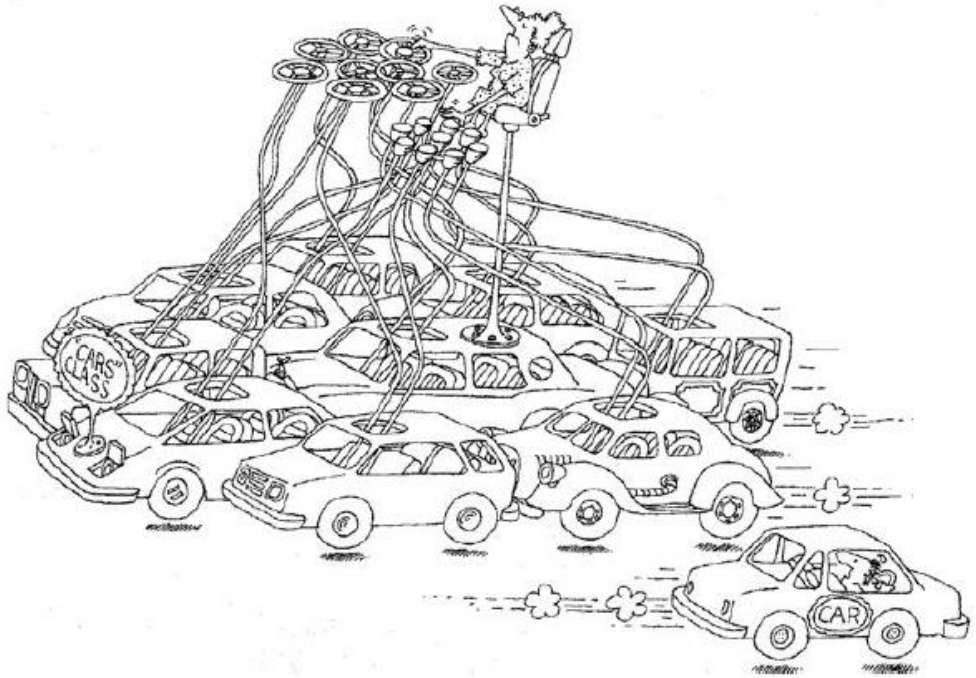
- The dot operator (.) allows you to access structure members in a program

FEW QUESTIONS

- What is Classification?
  - Minerals

CLASS

- A class represents a set of objects that share a common structure and common behavior.
- External Observable Attributes i.e. Unchanged



Class

- A *class* is a programmer-defined **data type**. It consists of **data** and **functions** which operate on that data.

- Placing data and functions together into a single entity is the **central idea** of object-oriented programming.

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.

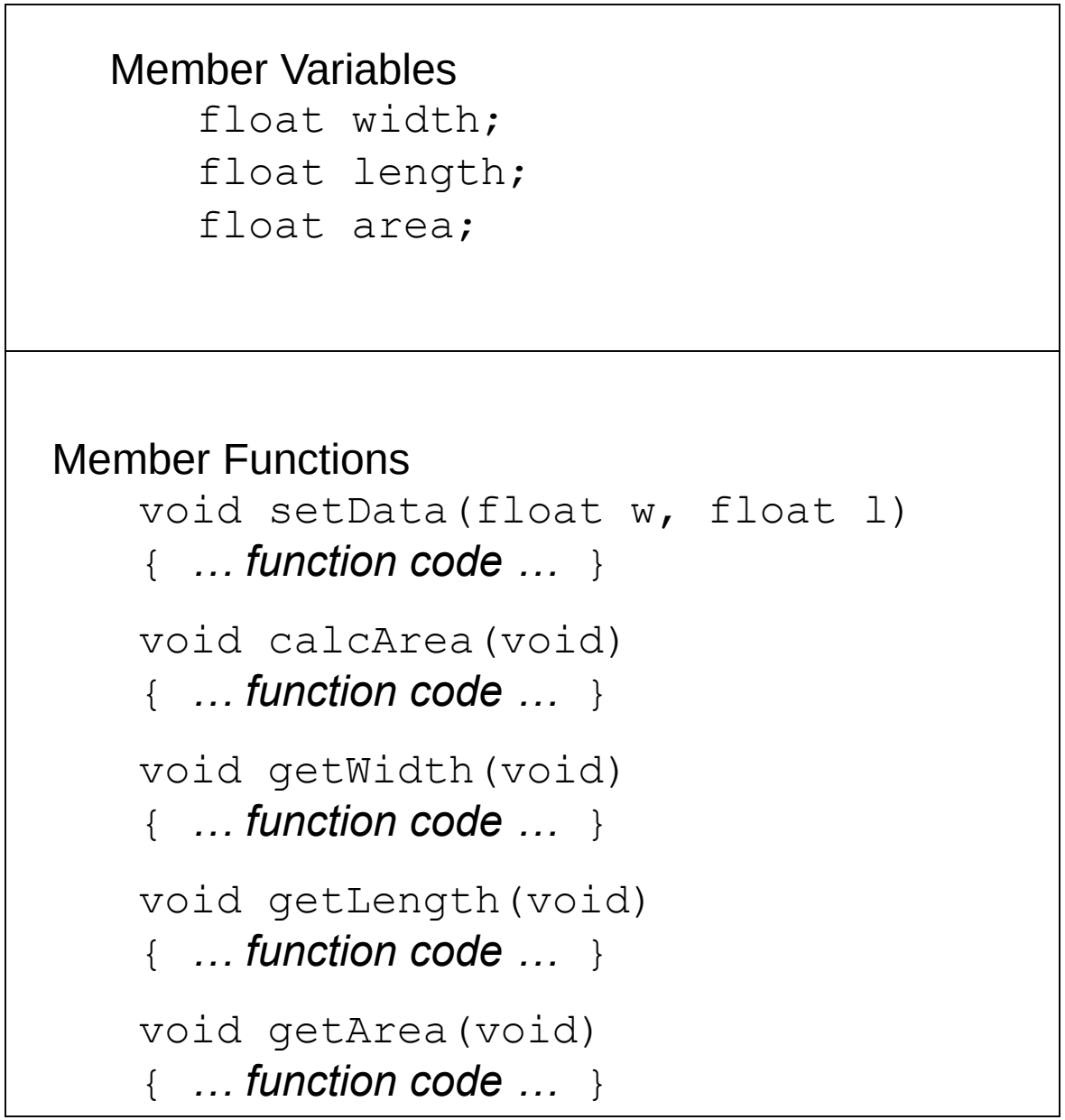
WHAT'S WRONG WITH PROCEDURAL PROGRAMMING?

- 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.

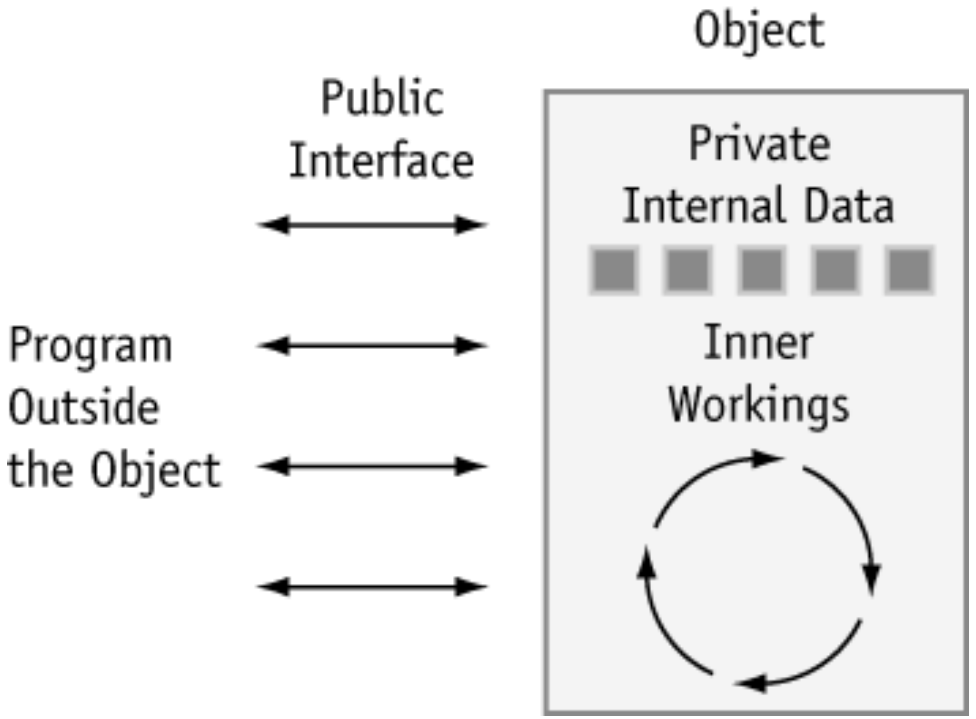
FIGURE 13-1


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();
```

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.

13.9 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
```