Overview

10.1 Structures
10.2 Classes
10.3 Abstract Data Types
10.4 Introduction to Inheritance
10.1 Structures
What Is a Class?

- A class is a data type whose variables are objects.
- Some pre-defined data types you have used are:
  - int
  - char
- A pre-defined class you have used is:
  - ifstream
- You can define your own classes as well.
Class Definitions

- A class definition includes
  - A description of the kinds of values the variable can hold
  - A description of the member functions
- We will start by defining structures as a first step toward defining classes
Structures

- A structure can be viewed as an object
  - Contains no member functions
    (The structures used here have no member functions)

- Contains multiple values of possibly different types
  - The multiple values are logically related as a single item
  - Example: A bank Certificate of Deposit (CD) has the following values:
    - a balance
    - an interest rate
    - a term (months to maturity)
The CD Definition

- The Certificate of Deposit structure can be defined as:

```c
struct CDAccount
{
    double balance;
    double interest_rate;
    int term;  //months to maturity
};
```

- Keyword struct begins a structure definition.
- CDAccount is the structure tag or the structure’s type.
- Member names are identifiers declared in the braces.

Remember this semicolon!
Using the Structure

- Structure definition is generally placed outside any function definition
  - This makes the structure type available to all code that follows the structure definition
- To declare two variables of type CDAccount:

  ```
  CDAccount  my_account, your_account;
  ```

- My_account and your_account contain distinct member variables balance, interest_rate, and term
The Structure Value

- The Structure Value
  - Consists of the values of the member variables

- The value of an object of type CDAccount
  - Consists of the values of the member variables
    - balance
    - interest_rate
    - term
Specifying Member Variables

- Member variables are specific to the structure variable in which they are declared.
  - Syntax to specify a member variable: Structure_Variable_Name . Member_Variable_Name
  - Given the declaration: CDAccount my_account, your_account;
    - Use the dot operator to specify a member variable
      my_account.balance
      my_account.interest_rate
      my_account.term
Using Member Variables

- Member variables can be used just as any other variable of the same type
  - my_account.balance = 1000;
  - your_account.balance = 2500;
  - Notice that my_account.balance and your_account.balance are different variables!
- my_account.balance = my_account.balance + interest;
Duplicate Names

- Member variable names duplicated between structure types are not a problem.

```
struct FertilizerStock
{
    double quantity;
    double nitrogen_content;
};

FertilizerStock super_grow;
```

```
struct CropYield
{
    int quantity;
    double size;
};

CropYield apples;
```

- `super_grow.quantity` and `apples.quantity` are different variables stored in different locations.
Structures as Arguments

- Structures can be arguments in function calls
  - The formal parameter can be call-by-value
  - The formal parameter can be call-by-reference
- Example:
  ```c
  void get_data(CDAccount& the_account);
  ```
  - Uses the structure type CDAccount we saw earlier as the type for a call-by-reference parameter
Structures can be the type of a value returned by a function

Example:

```c
CDAccount shrink_wrap(double the_balance,
                       double the_rate,
                       int the_term)
{
    CDAccount temp;
    temp.balance = the_balance;
    temp.interest_rate = the_rate;
    temp.term = the_term;
    return temp;
}
```
Using Function shrink_wrap

- shrink_wrap builds a complete structure value in temp, which is returned by the function.
- We can use shrink_wrap to give a variable of type CDAccount a value in this way:

```cpp
CDAccount new_account;
new_account = shrink_wrap(1000.00, 5.1, 11);
```
Assignment and Structures

- The assignment operator can be used to assign values to structure types

- Using the CDAccount structure again:
  ```
  CDAccount my_account, your_account;
  my_account.balance = 1000.00;
  my_account.interest_rate = 5.1;
  my_account.term = 12;
  your_account = my_account;
  ```

- Assigns all member variables in your_account the corresponding values in my_account
Hierarchical Structures

- Structures can contain member variables that are also structures

```c
struct Date {
    int month;
    int day;
    int year;
};
```

- `struct PersonInfo` contains a `Date` structure

```c
struct PersonInfo {
    double height;
    int weight;
    Date birthday;
};
```
Using PersonInfo

- A variable of type PersonInfo is declared by
  
  ```cpp
  PersonInfo person1;
  ```

- To display the birth year of person1, first access the birthday member of person1
  
  ```cpp
  cout << person1.birthday…
  ```

- But we want the year, so we now specify the year member of the birthday member
  
  ```cpp
  cout << person1.birthday.year;
  ```
Initializing Classes

- A structure can be initialized when declared
- Example:

  ```
  struct Date
  {
      int month;
      int day;
      int year;
  };
  ```

- Can be initialized in this way

  ```
  Date due_date = {12, 31, 2004};
  ```
Can you

- Write a definition for a structure type for records consisting of a person’s wage rate, accrued vacation (in whole days), and status (hourly or salaried). Represent the status as one of the two character values ‘H’ and ‘S’. Call the type EmployeeRecord.
10.2

Classes
Classes

- A class is a data type whose variables are objects
  - The definition of a class includes
    - Description of the kinds of values of the member variables
    - Description of the member functions
  - A class description is somewhat like a structure definition plus the member variables
A Class Example

- To create a new type named DayOfYear as a class definition
  - Decide on the values to represent
  - This example’s values are dates such as July 4 using an integer for the number of the month
    - Member variable month is an int (Jan = 1, Feb = 2, etc.)
    - Member variable day is an int
  - Decide on the member functions needed
  - We use just one member function named output
Class DayOfYear Definition

- class DayOfYear
  {
    public:
    void output();
    int month;
    int day;
  };
Defining a Member Function

- Member functions are declared in the class declaration
- Member function definitions identify the class in which the function is a member
  
  ```cpp
  void DayOfYear::output()
  {
    cout << "month = " << month
         << " , day = " << day
         << endl;
  }
  ```
Member Function Definition

- Member function definition syntax:
  Returned_Type  
  Class_Name::Function_Name( Parameter_List )
  {
    Function Body Statements
  }

- Example: void DayOfYear::output( )
  {
    cout << "month = " << month
        << ", day = " << day << endl;
  }
The ‘::’ Operator

- ‘::’ is the scope resolution operator
  - Tells the class a member function is a member of

- void DayOfYear::output() indicates that function output is a member of the DayOfYear class

- The class name that precedes ‘::’ is a type qualifier
‘::’ and ‘.’

- ‘::’ used with classes to identify a member
  ```
  void DayOfYear::output()
  {
    // function body
  }
  ```

- ‘.’ used with variables to identify a member
  ```
  DayOfYear birthday;
  birthday.output();
  ```
Calling Member Functions

- Calling the DayOfYear member function output is done in this way:
  
  ```cpp
  DayOfYear today, birthday;
  today.output( );
  birthday.output( );
  ```

- Note that today and birthday have their own versions of the month and day variables for use by the output function

Display 10.3 (1)
Display 10.3 (2)
Encapsulation

- Encapsulation is
  - Combining a number of items, such as variables and functions, into a single package such as an object of a class
Problems With DayOfYear

- Changing how the month is stored in the class DayOfYear requires changes to the program.
- If we decide to store the month as three characters (JAN, FEB, etc.) instead of an int:
  - `cin >> today.month` will no longer work because we now have three character variables to read.
  - `if(today.month == birthday.month)` will no longer work to compare months.
  - The member function “output” no longer works.
Ideal Class Definitions

- Changing the implementation of DayOfYear requires changes to the program that uses DayOfYear
- An ideal class definition of DayOfYear could be changed without requiring changes to the program that uses DayOfYear
Fixing DayOfYear

- To fix DayOfYear
  - We need to add member functions to use when changing or accessing the member variables
    - If the program never directly references the member variables, changing how the variables are stored will not require changing the program
  - We need to be sure that the program does not ever directly reference the member variables
Public Or Private?

- C++ helps us restrict the program from directly referencing member variables
  - private members of a class can only be referenced within the definitions of member functions
    - If the program tries to access a private member, the compiler gives an error message
  - Private members can be variables or functions
Private Variables

- Private variables cannot be accessed directly by the program
  - Changing their values requires the use of public member functions of the class
  - To set the private month and day variables in a new DayOfYear class use a member function such as

```cpp
void DayOfYear::set(int new_month, int new_day)
{
    month = new_month;
    day = new_day;
}
```
Public or Private Members

- The keyword private identifies the members of a class that can be accessed only by member functions of the class
  - Members that follow the keyword private are private members of the class
- The keyword public identifies the members of a class that can be accessed from outside the class
  - Members that follow the keyword public are public members of the class
A New DayOfYear

The new DayOfYear class demonstrated in Display 10.4...

- Uses all private member variables
- Uses member functions to do all manipulation of the private member variables
  - Member variables and member function definitions can be changed without changes to the program that uses DayOfYear
Using Private Variables

- It is normal to make all member variables private.
- Private variables require member functions to perform all changing and retrieving of values.
  - Accessor functions allow you to obtain the values of member variables.
    - Example: `get_day` in class `DayOfYear`.
  - Mutator functions allow you to change the values of member variables.
    - Example: `set` in class `DayOfYear`.
The syntax for a class definition is

```cpp
class Class_Name
{
    public:
        Member_Specification_1
        Member_Specification_2
        ...
        Member_Specification_3
    private:
        Member_Specification_n+1
        Member_Specification_n+2
        ...
};
```
Declaring an Object

- Once a class is defined, an object of the class is declared just as variables of any other type
  - Example: To create two objects of type Bicycle:
    - class Bicycle
      
      ```
      // class definition lines
      ```
    
      Bicycle my_bike, your_bike;
The Assignment Operator

- Objects and structures can be assigned values with the assignment operator (=)
  
  Example:

  ```
  DayOfYear due_date, tomorrow;
  tomorrow.set(11, 19);
  due_date = tomorrow;
  ```
Program Example: BankAccount Class

- This bank account class allows
  - Withdrawal of money at any time
  - All operations normally expected of a bank account (implemented with member functions)
  - Storing an account balance
  - Storing the account’s interest rate

Display 10.5 (1)
Display 10.5 (2)
Display 10.5 (3)
Display 10.5 (4)
Calling Public Members

- Recall that if calling a member function from the main function of a program, you must include the object name:

  account1.update();
Calling Private Members

- When a member function calls a private member function, an object name is not used
  - fraction (double percent);
    is a private member of the BankAccount class
  - fraction is called by member function update
    void BankAccount::update()
    {
        balance = balance +
                    fraction(interest_rate) * balance;
    }
Constructors

- A constructor can be used to initialize member variables when an object is declared
  - A constructor is a member function that is usually public
  - A constructor is automatically called when an object of the class is declared
  - A constructor’s name must be the name of the class
  - A constructor cannot return a value
    - No return type, not even void, is used in declaring or defining a constructor
A constructor for the BankAccount class could be declared as:

class BankAccount
{
    public:
        BankAccount(int dollars, int cents, double rate);
        //initializes the balance to $dollars.cents
        //initializes the interest rate to rate percent

        …//The rest of the BankAccount definition
};
Constructor Definition

- The constructor for the BankAccount class could be defined as

```cpp
BankAccount::BankAccount(int dollars, int cents, double rate)
{
    if ((dollars < 0) || (cents < 0) || (rate < 0))
    {
        cout << "Illegal values for money or rate\n";
        exit(1);
    }
    balance = dollars + 0.01 * cents;
    interest_rate = rate;
}
```

- Note that the class name and function name are the same
Calling A Constructor (1)

- A constructor is not called like a normal member function:

```c++
BankAccount account1;
account1.BankAccount(10, 50, 2.0);
```
A constructor is called in the object declaration

BankAccount account1(10, 50, 2.0);

- Creates a BankAccount object and calls the constructor to initialize the member variables
Overloading Constructors

- Constructors can be overloaded by defining constructors with different parameter lists
  - Other possible constructors for the BankAccount class might be

```java
BankAccount (double balance, double interest_rate);
BankAccount (double balance);
BankAccount (double interest_rate);
BankAccount ( );
```
The Default Constructor

- A default constructor uses no parameters
- A default constructor for the BankAccount class could be declared in this way
  ```cpp
  class BankAccount
  {
    public:
      BankAccount( ); // initializes balance to $0.00
      // initializes rate to 0.0%
      … // The rest of the class definition
  };
  ```
The default constructor for the BankAccount class could be defined as:

```cpp
BankAccount::BankAccount()
{
    balance = 0;
    rate = 0.0;
}
```

It is a good idea to always include a default constructor even if you do not want to initialize variables.
Calling the Default Constructor

- The default constructor is called during declaration of an object
  - An argument list is not used

```java
BankAccount account1;
// uses the default BankAccount constructor

BankAccount account1();
// Is not legal
```
Initialization Sections

- An initialization section in a function definition provides an alternative way to initialize member variables
  - BankAccount::BankAccount( ): balance(0),
    interest_rate(0.0);

{  
    // No code needed in this example
}

- The values in parenthesis are the initial values for the member variables listed
Parameters and Initialization

- Member functions with parameters can use initialization sections

```cpp
BankAccount::BankAccount(int dollars, int cents, double rate)
    : balance (dollars + 0.01 * cents),
      interest_rate(rate)
{
    if (( dollars < 0) || (cents < 0) || (rate < 0))
    {
        cout << "Illegal values for money or rate\n";
        exit(1);
    }
}
```

- Notice that the parameters can be arguments in the initialization
Member Initializers

- C++11 supports a feature called member initialization
  - Simply set member variables in the class
  - Ex: class Coordinate
    
    ```
    
    { 
        private:
            int x=1;
            int y=2;

        ...
    }
    ```

- Creating a Coordinate object will initialize its x variable to 1 and y to 2 (assuming a constructor isn’t called that sets the values to something else)
C++11 also supports constructor delegation. This lets you have a constructor invoke another constructor in the initialization section.

For example, make the default constructor call a second constructor that sets X to 99 and Y to 99:

```cpp
Coordinate::Coordinate() : Coordinate(99, 99)
{
}
```
Section 10.2 Conclusion

Can you

- Describe the difference between a class and a structure?
- Explain why member variables are usually private?
- Describe the purpose of a constructor?
- Use an initialization section in a function definition?
10.3

Abstract Data Types
Abstract Data Types

- A data type consists of a collection of values together with a set of basic operations defined on the values.
- A data type is an Abstract Data Type (ADT) if programmers using the type do not have access to the details of how the values and operations are implemented.
Classes To Produce ADTs

- To define a class so it is an ADT
  - Separate the specification of how the type is used by a programmer from the details of how the type is implemented
  - Make all member variables private members
  - Basic operations a programmer needs should be public member functions
  - Fully specify how to use each public function
  - Helper functions should be private members
ADT Interface

- The ADT interface tells how to use the ADT in a program
  - The interface consists of
    - The public member functions
    - The comments that explain how to use the functions
  - The interface should be all that is needed to know how to use the ADT in a program
ADT Implementation

- The ADT implementation tells how the interface is realized in C++
  - The implementation consists of
    - The private members of the class
    - The definitions of public and private member functions
- The implementation is needed to run a program
- The implementation is not needed to write the main part of a program or any non-member functions
ADT Benefits

- Changing an ADT implementation does require changing a program that uses the ADT
- ADT’s make it easier to divide work among different programmers
  - One or more can write the ADT
  - One or more can write code that uses the ADT
- Writing and using ADTs breaks the larger programming task into smaller tasks
In this version of the BankAccount ADT
- Data is stored as three member variables
  - The dollars part of the account balance
  - The cents part of the account balance
  - The interest rate
- This version stores the interest rate as a fraction
- The public portion of the class definition remains unchanged from the version of Display 10.6
Interface Preservation

- To preserve the interface of an ADT so that programs using it do not need to be changed
  - Public member declarations cannot be changed
  - Public member definitions can be changed
  - Private member functions can be added, deleted, or changed
Information Hiding

- Information hiding was referred to earlier as writing functions so they can be used like black boxes.
- ADT’s implement information hiding because:
  - The interface is all that is needed to use the ADT.
  - Implementation details of the ADT are not needed to know how to use the ADT.
  - Implementation details of the data values are not needed to know how to use the ADT.
Section 10.3 Conclusion

- Can you
  - Describe an ADT?
  - Describe how to implement an ADT in C++?
  - Define the interface of an ADT?
  - Define the implementation of an ADT?
10.4

Introduction to Inheritance
Inheritance

- Inheritance refers to derived classes
  - Derived classes are obtained from another class by adding features
  - A derived class inherits the member functions and variables from its parent class without having to rewrite them
- Example
  - In Chapter 6 we saw that the class of input-file streams is derived from the class of all input streams by adding member functions such as open and close
  - cin belongs to the class of all input streams, but not the class of input-file streams
Inheritance Example

- Natural hierarchy of bank accounts
- Most general: A Bank Account stores a balance
- A Checking Account “IS A” Bank Account that allows customers to write checks
- A Savings Account “IS A” Bank Account without checks but higher interest

Accounts are more specific as we go down the hierarchy

Each box can be a class
Inheritance Relationships

- The more specific class is a **derived** or **child** class
- The more general class is the **base**, **super**, or **parent** class
- If class B is derived from class A
  - Class B is a derived class of class A
  - Class B is a child of class A
  - Class A is the parent of class B
  - Class B inherits the member functions and variables of class A
Defining Derived Classes

- Give the class name as normal, but add a colon and then the name of the base class

```cpp
class SavingsAccount : public BankAccount
{
    ...
}
```

- Objects of type SavingsAccount can access member functions defined in SavingsAccount or BankAccount
Section 10.4 Conclusion

- Can you
  - Define object?
  - Define class?
  - Describe the relationship between parent and child classes?
  - Describe the benefit of inheritance?
Chapter 10 -- End
A Structure Definition (part 1 of 2)

```cpp
// Program to demonstrate the CDAccount structure type.
#include <iostream>
using namespace std;

// Structure for a bank certificate of deposit:
struct CDAccount
{
    double balance;
    double interest_rate;
    int term; // months until maturity
};

void get_data(CDAccount& the_account);
// Postcondition: the_account.balance and the_account.interest_rate
// have been given values that the user entered at the keyboard.

int main()
{
    CDAccount account;
    get_data(account);

    double rate_fraction, interest;
    rate_fraction = account.interest_rate/100.0;
    interest = account.balance*rate_fraction*(account.term/12.0);
    account.balance = account.balance + interest;

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "When your CD matures in "
         << account.term << " months,\n         it will have a balance of $"
         << account.balance << endl;
    return 0;
}
```
A Structure Definition (part 2 of 2)

```cpp
//Uses iostream:
void get_data(CDAccount& the_account)
{
    cout << "Enter account balance: ";
    cin >> the_account.balance;
    cout << "Enter account interest rate: ";
    cin >> the_account.interest_rate;
    cout << "Enter the number of months until maturity\n" << "(must be 12 or fewer months): ";
    cin >> the_account.term;
}
```

Sample Dialogue

Enter account balance: $100.00
Enter account interest rate: 10.0
Enter the number of months until maturity
(must be 12 or fewer months): 6
When your CD matures in 6 months, it will have a balance of $105.00
Member Values

```c
struct CDAccount
{
    double balance;
    double interest_rate;
    int term; // months until maturity
};
int main()
{
    CDAccount account;
    ...  
    account.balance = 1000.00;
    account.interest_rate = 4.7;
    account.term = 11;
    balance ? account
    interest_rate ? account
    term ? account
    balance 1000.00 account
    interest_rate 4.7 account
    term 11 account
```
DISPLAY 10.3  Class with a Member Function (part 1 of 2)

```cpp
//Program to demonstrate a very simple example of a class.
//A better version of the class DayOfYear will be given in Display 10.4.
#include <iostream>
using namespace std;

class DayOfYear
{
  public:
    void output();  // Member function declaration
    int month;
    int day;
};

int main()
{
  DayOfYear today, birthday;
  cout << "Enter today's date:\n";
  cout << "Enter month as a number: ";
  cin >> today.month;
  cout << "Enter the day of the month: ";
  cin >> today.day;
  cout << "Enter your birthday:\n";
  cout << "Enter month as a number: ";
  cin >> birthday.month;
  cout << "Enter the day of the month: ";
  cin >> birthday.day;
  cout << "Today's date is ";
  today.output();
  cout << "Your birthday is ";
  birthday.output();
  if (today.month == birthday.month
    && today.day == birthday.day)
    cout << "Happy Birthday!\n";
  else
    cout << "Happy Unbirthday!\n";
  return 0;
}

//Uses iostream:
void DayOfYear::output()
{
  cout << "month = " << month
       << " , day = " << day << endl;
}
```

(continued)
Display 10.3
(2/2)

**DISPLAY 10.3**  Class with a Member Function *(part 2 of 2)*

---

**Sample Dialogue**

Enter today's date:
Enter month as a number: 10
Enter the day of the month: 15
Enter your birthday:
Enter month as a number: 2
Enter the day of the month: 21
Today's date is month = 10, day = 15
Your birthday is month = 2, day = 21
Happy Unbirthday!
DISPLAY 10.4  Class with Private Members (part 1 of 2)

```cpp
// Program to demonstrate the class DayOfYear.
#include <iostream>
using namespace std;

class DayOfYear
{
  public:
    void input();
    void output();
    void set(int new_month, int new_day);
    // Precondition: new_month and new_day form a possible date.
    // Postcondition: The date is reset according to the arguments.
    int get_month();
    // Returns the month, 1 for January, 2 for February, etc.
    int get_day();
    // Returns the day of the month.
  private:
    void check_date();
    int month;
    int day;
};

int main()
{
  DayOfYear today, bach_birthday;
  cout << "Enter today's date:\n";
  today.input();
  cout << "Today's date is ";
  today.output();
  bach_birthday.set(3, 21);
  cout << "J. S. Bach's birthday is ";
  bach_birthday.output();
  if (today.get_month() == bach_birthday.get_month() &&
      today.get_day() == bach_birthday.get_day())
    cout << "Happy Birthday Johann Sebastian!\n";
  else
    cout << "Happy Unbirthday Johann Sebastian!\n";
  return 0;
}
```

(continued)
DISPLAY 10.4  Class with Private Members (part 2 of 2)

42 cin >> month;
43 cout << "Enter the day of the month: ";
44 cin >> day;
45 check_date();
46 }
47
48 void DayOfYear::output()
49 {
50   <The rest of the definition of DayOfYear::output is given in Display 10.3.>
51 }
52
53 void DayOfYear::set(int new_month, int new_day)
54 {
55   month = new_month;
56   day = new_day;
57   check_date();
58 }
59
do
60   61 void DayOfYear::check_date()
62 {
63     if ((month < 1) || (month > 12) || (day < 1) || (day > 31))
64     {
65       cout << "Illegal date. Aborting program.\n";
66       exit(1);
67     }
68 }  // end check_date()
69
70 int DayOfYear::get_month()
71 {
72   return month;
73 }
74
75 int DayOfYear::get_day()
76 {
77   return day;
78 }
79

Sample Dialogue

Enter today's date:
Enter the month as a number: 3
Enter the day of the month: 21
Today's date is month = 3, day = 21
J. S. Bach's birthday is month = 3, day = 21
Happy Birthday Johann Sebastian!
The BankAccount Class (part 1 of 4)

// Program to demonstrate the class BankAccount.
#include <iostream>
using namespace std;

// Class for a bank account:
class BankAccount
{
    public:
        void set(int dollars, int cents, double rate);
        // Postcondition: The account balance has been set to $dollars.cents;
        // The interest rate has been set to rate percent.
        void set(int dollars, double rate);
        // Postcondition: The account balance has been set to $dollars.00.
        // The interest rate has been set to rate percent.
        void update();
        // Postcondition: One year of simple interest has been
        // added to the account balance.
        double get_balance();
        // Returns the current account balance.
        double get_rate();
        // Returns the current account interest rate as a percentage.
        void output(ostream& outs);
        // Precondition: If outs is a file output stream, then
        // outs has already been connected to a file.
        // Postcondition: Account balance and interest rate have been written to the
        // stream outs.
    private:
        double balance;
        double interest_rate;

        double fraction(double percent);
        // Converts a percentage to a fraction. For example, fraction(50.3) returns 0.503.
    };

    int main()
    {
        BankAccount account1, account2;
        cout << "Start of Test:\n";
The BankAccount Class (part 2 of 4)

```c++
account1.set(123, 99, 3.0);
cout << "account1 initial statement:\n";
account1.output(cout);

account1.set(100, 5.0);
cout << "account1 with new setup: \n";
account1.output(cout);

account1.update();
cout << "account1 after update: \n";
account1.output(cout);

account2 = account1;
cout << "account2:\n";
account2.output(cout);
return 0;
}

void BankAccount::set(int dollars, int cents, double rate)
{
if ((dollars < 0) || (cents < 0) || (rate < 0))
{
cout << "Illegal values for money or interest rate.\n"
exit(1);
}
balance = dollars + 0.01*cents;
interest_rate = rate;
}

void BankAccount::set(int dollars, double rate)
{
if ((dollars < 0) || (rate < 0))
{
cout << "Illegal values for money or interest rate.\n"
exit(1);
}
balance = dollars;
interest_rate = rate;
}```
The BankAccount Class (part 3 of 4)

```cpp
void BankAccount::update()
{
    balance = balance + fraction(interest_rate)*balance;
}

double BankAccount::fraction(double percent_value)
{
    return (percent_value/100.0);
}

double BankAccount::get_balance()
{
    return balance;
}

double BankAccount::get_rate()
{
    return interest_rate;
}

//Uses iosstream:
void BankAccount::output(ostream& outs)
{
    outs.setf(ios::fixed);
    outs.setf(ios::showpoint);
    outs.precision(2);
    outs << "Account balance $" << balance << endl;
    outs << "Interest rate " << interest_rate << "\%" << endl;
}
```

In the definition of a member function, you call another member function like this.

Stream parameter that can be replaced with either cout or with a file output stream.
The BankAccount Class (part 4 of 4)

Sample Dialogue

Start of Test:
account1 initial statement:
Account balance $123.99
Interest rate 3.00%
account1 with new setup:
Account balance $100.00
Interest rate 5.00%
account1 after update:
Account balance $105.00
Interest rate 5.00%
account2:
Account balance $105.00
Interest rate 5.00%
DISPLAY 10.6  Class with Constructors (part 1 of 3)

1   //Program to demonstrate the class BankAccount.
2   #include <iostream>
3       using namespace std;
4
5   //Class for a bank account:
6   class BankAccount
7   {
8     public:
9       BankAccount(int dollars, int cents, double rate);
10      //Initializes the account balance to $dollars.cents and
11      //initializes the interest rate to rate percent.
12       BankAccount(int dollars, double rate);
13      //Initializes the account balance to $dollars.00 and
14      //initializes the interest rate to rate percent.
15       BankAccount();  //Default constructor
16      //Initializes the account balance to $0.00 and the interest rate to 0.0%.

(continued)
DISPLAY 10.6  Class with Constructors (part 2 of 3)

16          void update();
17          //Postcondition: One year of simple interest has been added to the account
18          //balance.
19          double get_balance();
20          //Returns the current account balance.
21          double get_rate();
22          //Returns the current account interest rate as a percentage.
23          void output(ostream& outs);
24          //Precondition: If outs is a file output stream, then
25          //outs has already been connected to a file.
26          //Postcondition: Account balance and interest rate have been written to the
27          //stream outs.
28          private:
29          double balance;
30          double interest_rate;
31          double fraction(double percent);
32          //Converts a percentage to a fraction. For example, fraction(50.3)
33          //returns 0.503.
34          
35          int main()
36          {
37          BankAccount account1(100, 2.3), account2;
38          cout << "account1 initialized as follows:\n";
39          account1.output(cout);
40          cout << "account2 initialized as follows:\n";
41          account2.output(cout);
42          account1 = BankAccount(999, 99, 5.5);
43          cout << "account1 reset to the following:\n";
44          account1.output(cout);
45          return 0;
46          }
47          BankAccount::BankAccount(int dollars, int cents, double rate)
48          {
49          if ((dollars < 0) || (cents < 0) || (rate < 0))
50          {
51          cout << "Illegal values for money or interest rate.\n";
52          exit(1);
53          }
54          (continued)
Display 10.6  Class with Constructors (part 3 of 3)

```cpp
56       balance = dollars + 0.01*cents;
57       interest_rate = rate;
58     }
59
60     BankAccount::BankAccount(int dollars, double rate)
61     {
62         if ((dollars < 0) || (rate < 0))
63             {  
64                 cout << "Illegal values for money or interest rate.\n";
65                 exit(1);
66             }
67         balance = dollars;
68         interest_rate = rate;
69     }
70
71     BankAccount::BankAccount() : balance(0), interest_rate(0.0)
72     {  
73         //Body intentionally empty    <Definitions of the other member functions
74     }
```

**Screen Output**

account1 initialized as follows:
Account balance $100.00
Interest rate 2.30%

account2 initialized as follows:
Account balance $0.00
Interest rate 0.00%

account1 reset to the following:
Account balance $999.99
Interest rate 5.50%
DISPLAY 10.7 Alternative BankAccount Class Implementation (part 1 of 3)

```cpp
// Demonstrates an alternative implementation of the class BankAccount.
#include <iostream>
#include <cmath>
using namespace std;

// Class for a bank account: Notice that the public members of BankAccount look and behave exactly the same as in Display 10.6.
class BankAccount
{
public:
    BankAccount(int dollars, int cents, double rate);
    // Initializes the account balance to $dollars.cents and initializes the interest rate to rate percent.
    BankAccount(int dollars, double rate);
    // Initializes the account balance to $dollars.00 and initializes the interest rate to rate percent.
    BankAccount();
    // Initializes the account balance to $0.00 and the interest rate to 0.0%.
    void update();
    // Postcondition: One year of simple interest has been added to the account balance.
    double get_balance();
    // Returns the current account balance.
    double get_rate();
    // Returns the current account interest rate as a percentage.
    void output(ostream& outs);
    // Precondition: If outs is a file output stream, then outs has already been connected to a file.
    // Postcondition: Account balance and interest rate have been written to the stream outs.

private:
    int dollars_part;
    int cents_part;
    double interest_rate; // expressed as a fraction, for example, 0.057 for 5.7
    double fraction(double percent);
    // Converts a percentage to a fraction. For example, fraction(50.3) returns 0.503.
    double percent(double fraction_value);
    // Converts a fraction to a percentage. For example, percent(0.503) returns 50.3.
};
(continued)
```
DISPLAY 10.7  Alternative BankAccount Class Implementation (part 2 of 3)

```cpp
int main()
{
    BankAccount account1(100, 2.3), account2;
    cout << "account1 initialized as follows:\n";
    account1.output(cout);
    cout << "account2 initialized as follows:\n";
    account2.output(cout);
    account1 = BankAccount(999, 99, 5.5);
    cout << "account1 reset to the following:\n";
    account1.output(cout);
    return 0;
}

BankAccount::BankAccount(int dollars, int cents, double rate)
{
    if ((dollars < 0) || (cents < 0) || (rate < 0))
    {
        cout << "Illegal values for money or interest rate.\n";
        exit(1);
    }
    dollars_part = dollars;
    cents_part = cents;
    interest_rate = fraction(rate);
}

BankAccount::BankAccount(int dollars, double rate)
{
    if ((dollars < 0) || (rate < 0))
    {
        cout << "Illegal values for money or interest rate.\n";
        exit(1);
    }
    dollars_part = dollars;
    cents_part = 0;
    interest_rate = fraction(rate);
}

BankAccount::BankAccount() : dollars_part(0), cents_part(0), interest_rate(0.0)
{
    //Body intentionally empty.
}
```

(continued)
DISPLAY 10.7 Alternative BankAccount Class Implementation (part 3 of 3)

```cpp
84 double BankAccount::fraction(double percent_value)
85 {
86   return (percent_value/100.0);
87 }
88
89 //Uses cmath:
90 void BankAccount::update()
91 {
92   double balance = get_balance();
93   balance = balance + interest_rate*balance;
94   dollars_part = floor(balance);
95   cents_part = floor((balance - dollars_part)*100);
96 }
97
98 double BankAccount::get_balance()
99 {
100   return (dollars_part + 0.01*cents_part);
101 }
102
103 double BankAccount::percent(double fraction_value)
104 {
105   return (fraction_value*100);
106 }
107
108 double BankAccount::get_rate()
109 {
110   return percent(interest_rate);
111 }
112
113 //Uses iostream:
114 void BankAccount::output(ostream& outs)
115 {
116   outs.setf(ios::fixed);
117   outs.setf(ios::showpoint);
118   outs.precision(2);
119   outs << "Account balance $" << get_balance() << endl;
120   outs << "Interest rate " << get_rate() << "%" << endl;
121 }
```

The new definitions of `get_balance` and `get_rate` ensure that the output will still be in the correct units.
Display 10.8

A Class Hierarchy

- Bank Account
  - Checking Account
  - Money Market Account
  - Savings Account
  - CD Account
class SavingsAccount : public BankAccount {
  
public:
  
    SavingsAccount(int dollars, int cents, double rate);
    // Other constructors would go here
    void deposit(int dollars, int cents);
    // Adds $dollars.cents to the account balance
    void withdraw(int dollars, int cents);
    // Subtracts $dollars.cents from the account balance
  
private:
};

int main() {
  
    SavingsAccount account(100, 50, 5.5);
    account.output(cout);
    cout << endl;
    cout << "Depositing $10.25." << endl;
    account.deposit(10,25);
    account.output(cout);
    cout << endl;
    cout << "Withdrawing $11.80." << endl;
    account.withdraw(11,80);
    account.output(cout);
    cout << endl;
    return 0;
}
The `SavingsAccount` constructor invokes the `BankAccount` constructor. Note the preceding colon.

```cpp
SavingsAccount::SavingsAccount(int dollars, int cents, double rate):
   BankAccount(dollars, cents, rate)
{
   // deliberately empty
}

void SavingsAccount::deposit(int dollars, int cents)
{
   double balance = get_balance();
   balance += dollars;
   balance += (static_cast<double>(cents) / 100);  
   int new_dollars = static_cast<int>(balance);
   int new_cents = static_cast<int>((balance - new_dollars) * 100);
```
set(new_dollars, new_cents, get_rate());

```cpp
void SavingsAccount::withdraw(int dollars, int cents)
{
    double balance = get_balance();
    balance -= dollars;
    balance -= (static_cast<double>(cents) / 100);
    int new_dollars = static_cast<int>(balance);
    int new_cents = static_cast<int>((balance - new_dollars) * 100);
    set(new_dollars, new_cents, get_rate());
}
```

**Screen Output**

Account balance $100.50
Interest rate 5.50%
Depositing $10.25.
Account balance $110.75
Interest rate 5.50%
Withdrawing $11.80.
Account balance $98.95
Interest rate 5.50%