Classes and Objects
Classes and Objects

- Class Definitions and Objects
- Member Functions
- Data Members
  - Get and Set functions
  - Constructors
- Placing Classes in Separate Files
- Separating Interface from Implementation
- Data Validation
  - Ensures that data in an object is in a particular format or range.
C++ Program Structure

- Typically C++ Programs will consist of:
  - A function **main**
  - One or more classes
    - Each containing **data members** and **member functions**.
16.2 Defining a Class
With a Member Function

• Class definition
  - Tells the compiler what **member functions** and **data members** belong to the class.
  - Keyword **class** followed by the class’s name.
  - Class body is enclosed in braces ({}).
    • Specifies data members and member functions
    • Access-specifier **public**: Indicates that a member function or data member is accessible to other functions and member functions of other classes.
// Fig. 19.1: fig19_01.cpp
// Define class GradeBook with a member function displayMessage;
// Create a GradeBook object and call its displayMessage function.
#include <iostream>
using std::cout;
using std::endl;

// GradeBook class definition
class GradeBook
{
public:
    // function that displays a welcome message
    void displayMessage()
    {
        cout << "Welcome to the Grade Book!" << endl;
    }
}; // end class GradeBook

// function main begins program execution
int main()
{
    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
    myGradeBook.displayMessage(); // call object's displayMessage function
    return 0; // indicate successful termination
} // end main

Welcome to the Grade Book!
// Fig. 19.3: fig19_03.cpp
// Define class GradeBook with a member function that takes a parameter;
// Create a GradeBook object and call its displayMessage function.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <string> // program uses C++ standard string class
using std::string;
using std::getline;

// GradeBook class definition
class GradeBook
{
public:
    // function that displays a welcome message to the GradeBook user
    void displayMessage(string courseName)
    {
        cout << "Welcome to the grade book for\n" << courseName << "!" << endl;
    }
}; // end class GradeBook

// function main begins program execution
int main()
{
    string nameOfCourse; // string of characters to store the course name
    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
// prompt for and input course name
cout << "Please enter the course name:" << endl;
getline( cin, nameOfCourse ); // read a course name with blanks

// call myGradeBook's displayMessage function
// and pass nameOfCourse as an argument
myGradeBook.displayMessage( nameOfCourse );

return 0; // indicate successful termination

Please enter the course name: 
CS101 Introduction to C++ Programming

Welcome to the grade book for
CS101 Introduction to C++ Programming!

getline is a library fcn
Passing an argument to the member function
Member Function Takes a Parameter

- A string
  - Represents a string of characters.
  - An object of C++ Standard Library class `std::string`.
    - Defined in header file `<string>`.

- Library function `getline`
  - Used to retrieve input until a newline is encountered.
  - Example
    - `getline( cin, nameOfCourse );`
      - Inputs a line from standard input into string object `nameOfCourse`. 

© 2007 Pearson Ed - All rights reserved.
Local variables
- Variables declared in a function definition’s body cannot be used outside of that function body.
- When a function terminates the values of its local variables are lost.

Attributes
- Exist throughout the life of the object.
- Are represented as data members.
  - Namely, associated with variables in a class definition.
  - Are declared inside a class definition but outside the bodies of the class’s member-function definitions.
- Each object of a class maintains its own copy of its attributes in memory.
16.4 Data Members, set Functions and get Functions

- Access-specifier private
  - Makes a data member or member function accessible only to member functions of the class.
  - private is the default access for class members.
  - “information hiding” is an object-oriented tenet.

- Returning a value from a function
  - A function that specifies a return type other than void
    - Returns a value to its calling function.
// Fig. 19.5: fig19_05.cpp
// Define class GradeBook that contains a courseName data member
// and member functions to set and get its value;
// Create and manipulate a GradeBook object with these functions.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <string> // program uses C++ standard string class
using std::string;
using std::getline;

// GradeBook class definition
class GradeBook
{
public:

    // function that sets the course name
void setCourseName(string name)
    {
        courseName = name; // store the course name in the object
    } // end function setCourseName

    // function that gets the course name
string getCourseName()
    {
        return courseName; // return the object's courseName
    } // end function getCourseName
}; // end class GradeBook

set function modifies private data
get function accesses private data
// function that displays a welcome message
void displayMessage()
{
    // this statement calls getCourseName to get the
    // name of the course this GradeBook represents
    cout << "Welcome to the grade book for\n" << getCourseName() << "!"
        << endl;
} // end function displayMessage

private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook

// function main begins program execution
int main()
{
    string nameOfCourse; // string of characters to store the course name
    GradeBook myGradeBook; // create a GradeBook object named myGradeBook

    // display initial value of courseName
    cout << "Initial course name is: " << myGradeBook.getCourseName()
        << endl;

    // use set and get functions, even within the class
    // private members accessible only to member functions of the class
    // default constructor
    // accessing private data outside class definition

© 2007 Pearson Ed - All rights reserved.
// prompt for, input and set course name
cout << "\nPlease enter the course name:" << endl;
getline( cin, nameOfCourse ); // read a course name with blanks
myGradeBook.setCourseName( nameOfCourse ); // set the course name

// outputs a blank line
cout << endl;
myGradeBook.displayMessage(); // display message with new course name
return 0; // indicate successful termination

Initial course name is:
Please enter the course name:
CS101 Introduction to C++ Programming
Welcome to the grade book for
CS101 Introduction to C++ Programming!

Modifying private data outside class definition

default setting from constructor is an empty string!!
As a rule of thumb, data members should be declared private and member functions should be declared public. (We will see that it is appropriate to declare certain member functions private, if they are to be accessed only by other member functions of the class.)
Data Members, **set** Functions and **get** Functions

- Software engineering with **set** and **get** functions:
  - **public** member functions that allow clients of a class to **set** or **get** the values of **private** data members.
  - **set** functions are sometimes called **mutators** and **get** functions are sometimes called **accessors**.
  - Allows the creator of the class to control how clients access **private** data.
  - Should also be used by other member functions of the same class.
Constructors

- Functions used to initialize an object's data when it is created.
  - The call is made \textit{implicitly} by the compiler when the object is created.
  - Must be defined with the same name as the class.
  - Cannot return values.
    - Not even \texttt{void}!!
  - A \texttt{default constructor} has no parameters.
    - The compiler will provide one when a class does not explicitly include a constructor.
    - A compiler's default constructor only calls constructors of data members that are objects of classes.
16.5 Initializing Objects with Constructors

- Any constructor that takes no arguments is called a **default constructor**.

- A class gets a default constructor in one of two ways:
  - The compiler implicitly creates a default constructor in a class that does not define a constructor. Such a constructor does not initialize the class’s data members, but does call the default constructor for each data member that is an object of another class. An uninitialized variable typically contains a “garbage” value.
  
  - You explicitly define a constructor that takes no arguments. Such a default constructor will call the default constructor for each data member that is an object of another class and will perform additional initialization specified by you.

- If you define a constructor with arguments, C++ will not implicitly create a default constructor for that class.
Constructor Example

Constructor has same name as class and no return type

```
#include <iostream>
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
    // constructor initializes courseName with string supplied as argument
    GradeBook( string name )
    {
        setCourseName( name ); // call set function to initialize courseName
    } // end GradeBook constructor
```

Fig. 3.7 | Instantiating multiple objects of the GradeBook class and using the GradeBook constructor to specify the course name when each GradeBook object is created. (Part 1 of 3.)
Constructor Example

```c++
19    // function to set the course name
20 void setCourseName( string name )
21 {
22      courseName = name; // store the course name in the object
23 } // end function setCourseName
24
25    // function to get the course name
26 string getCourseName()
27 {
28      return courseName; // return object's courseName
29 } // end function getCourseName
30
31    // display a welcome message to the GradeBook user
32 void displayMessage()
33 {
34      // call getCourseName to get the courseName
35      cout << "Welcome to the grade book for" << getCourseName()
36         << "!" << endl;
37     } // end function displayMessage
```

**Fig. 3.7** | Instantiating multiple objects of the GradeBook class and using the GradeBook constructor to specify the course name when each GradeBook object is created. (Part 2 of 3.)
Constructor Example

```cpp
private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook

// function main begins program execution
int main()
{
    // create two GradeBook objects
    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
    GradeBook gradeBook2( "CS102 Data Structures in C++" );

    // display initial value of courseName for each GradeBook
    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
         << "gradeBook2 created for course: " << gradeBook2.getCourseName()
         << endl;
    } // end main
```

Fig. 3.7 | Instantiating multiple objects of the GradeBook class and using the GradeBook constructor to specify the course name when each GradeBook object is created. (Part 3 of 3.)
Placing a Class in a Separate File for Reusability

- **.cpp** file is known as a source-code file.
- Header files
  - Separate files in which class definitions are placed.
  - Allows compiler to recognize the classes when used elsewhere.
  - Generally have `.h` filename extensions
- Driver files
  - A program used to test software (such as classes).
  - Contains a `main` function so it can be executed.
16.7 Separating Interface from Implementation

- **Interface**
  - Describes what services a class’s clients can use and how to request those services.
    - without revealing how the class carries out the services.
    - a class definition that lists only member function names, return types and parameter types.
      - e.g., function prototypes
    - A class’s interface consists of the class’s **public** member functions (services).
  
- **Separating interface from implementation:**
  - Client code should not break if implementation changes, as long as the interface stays the same.
Separating Interface from Implementation

- Define the member functions outside the class definition, in a separate source-code file.
  - In a source-code file for a class
    - Use **binary scope resolution operator** (:::) to tie each member function to the class definition.
  - Implementation details are hidden.
    - Client code does not need to know the implementation.
- In a header file for a class
  - The function prototypes describe the class’s **public** interface.
// Fig. 19.11: GradeBook.h
// GradeBook class definition. This file presents GradeBook's public
// interface without revealing the implementations of GradeBook's member
// functions, which are defined in GradeBook.cpp.
#include <string> // class GradeBook uses C++ standard string class
using std::string;

// GradeBook class definition
class GradeBook
{
public:
    GradeBook( string ); // constructor that initializes courseName
    void setCourseName( string ); // function that sets the course name
    string getCourseName(); // function that gets the course name
    void displayMessage(); // function that displays a welcome message

private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook
// Fig. 19.12: GradeBook.cpp
// GradeBook member-function definitions. This file contains
// implementations of the member functions prototyped in GradeBook.h.
#include <iostream>
using std::cout;
using std::endl;

#include "GradeBook.h" // include definition of class GradeBook

// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name )
{
    setCourseName( name ); // call set function to initialize courseName
} // end GradeBook constructor

// function to set the course name
void GradeBook::SetCourseName( string name )
{
    courseName = name; // store the course name in the object
} // end function SetCourseName
// function to get the course name
string GradeBook::getCourseName()
{
    return courseName; // return object's courseName
} // end function getCourseName

// display a welcome message to the GradeBook user
void GradeBook::displayMessage()
{
    // call getCourseName to get the courseName
    cout << "Welcome to the grade book for\n" << getCourseName()
         << "!" << endl;
} // end function displayMessage
// Fig. 19.13: fig19_13.cpp
// GradeBook class demonstration after separating
// its interface from its implementation.
#include <iostream>
using std::cout;
using std::endl;

#include "GradeBook.h" // include definition of class GradeBook

// function main begins program execution
int main()
{
    // create two GradeBook objects
    GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
    GradeBook gradeBook2( "CS102 Data Structures in C++" );

    // display initial value of courseName for each GradeBook
    cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
        << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
        << endl;
    return 0; // indicate successful termination
} // end main

Note - This is a separate .cpp file that holds only main.
19.10 Validating Data with set Functions

- **set** functions can validate data.
  - Known as validity checking.
  - Keeps object in a consistent state.
    - The data member contains a valid value.
  - Can return values indicating that attempts were made to assign invalid data.

- **string** member functions
  - **length** returns the number of characters in the string.
  - **substr** returns specified substring within the string.
Validating Data with \texttt{set} Functions

\begin{verbatim}
// Fig. 19.16: GradeBook.cpp
// Implementations of the GradeBook member-function definitions.
// The \texttt{setCourseName} function performs validation.
#include <iostream>
using std::cout;
using std::endl;

#include "GradeBook.h" // include definition of class GradeBook

// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name )
{
    setCourseName( name ); // validate and store courseName
} // end GradeBook constructor

// function that sets the course name;
// ensures that the course name has at most 25 characters
void GradeBook::setCourseName( string name )
{
    if ( name.length() <= 25 ) // if name has 25 or fewer characters
        courseName = name; // store the course name in the object
}
\end{verbatim}
Validating Data with \textit{set} Functions

```cpp
if (name.length() > 25) // if name has more than 25 characters
{
  // set courseName to first 25 characters of parameter name
  courseName = name.substr(0, 25); // start at 0, length of 25
  cout << "Name" " " name << " exceeds maximum length (25)." "
  "Limiting courseName to first 25 characters." " << endl;
} // end if

// function to get the course name
string GradeBook::getCourseName()
{
  return courseName; // return object's courseName
} // end function getCourseName

// display a welcome message to the GradeBook user
void GradeBook::displayMessage()
{
  // call getCourseName to get the courseName
  cout << "Welcome to the grade book for" " getCourseName()" "!
  " << endl;
} // end function displayMessage
```

© 2007 Pearson Ed - All rights reserved.
Review of Classes and Objects

- Introduced **class definitions and objects**
  - Public versus private access into class.
- **Syntax for member functions**
- **Syntax data members**
  - Get and Set functions
  - Constructors
- Placing classes in separate files
- Separating interface from implementation
- Data validation in set functions.