Outline

C++ Classes and Objects
  - Classes
  - Objects
  - Member Functions
  - Data Members
  - Initializing Objects with Constructors
  - Separating Interface from Implementation
  - Validating Data

Q&A
C++ Programs

- Function `main`
- One or more classes
  - Each containing data members and member functions
Classes, Objects, Member Functions, & Data Members

- Classes: Car example
  - Functions describe the mechanisms that perform tasks, such as acceleration
    - Hide complex tasks from the user, just as a driver can use the pedal to accelerate without needing to know how the acceleration is performed
Classes, Objects, Member Functions, & Data Members Cont’d

- Classes: Car example Cont’d
  - Classes must be defined before they can be used; a car must be built before it can be driven
  - Many car objects can be created from the same class, many cars can be built from same engineering drawing
Member-function calls send messages to an object to perform tasks, just like pressing the gas pedal sends a message to the car to accelerate.

Objects and cars both have attributes, like color and miles driven.
Class with a Member Function

- **Class definition**
  - Tells 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. 3.1: fig03_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 to the user
    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!
Common Programming Error 1

Forgetting the semicolon at the end of a class definition is a syntax error.
Member function definition

- Return type of a function
  - Indicates the type of value returned by the function when it completes its task
  - `void` indicates that the function does not return any value

- Function name must be a valid identifier

- Parenteses after function name indicate that it is a function

- Function body contains statements that perform the function’s task
  - Delimited by braces (`{}`)
Common Programming Error 2

Returning a value from a function whose return type has been declared void is a compilation error.
Common Programming Error 3

Defining a function inside another function is a syntax error.
Class w/ a Member Function Cont’d

- **Using a class**
  - A class is a user-defined type (or programmer-defined type)
    - Can be used to create objects
      - Variables of the class type
    - C++ is an extensible language
  - Dot operator (.)
    - Used to access an object’s data members and member functions
    - Example
      - `myGradeBook.displayMessage()`
        - Call member function `displayMessage` of `GradeBook` object `myGradeBook`
Member Function w/ a Parameter

- **Function parameter(s)**
  - Information needed by a function to perform its task

- **Function argument(s)**
  - Values supplied by a function call for each of the function’s parameters
    - Argument values are copied into function parameters at execution time
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 newline is encountered

Example

```cpp
getline( cin, nameOfCourse );
```

- Inputs a line from standard input into string object `nameOfCourse`
// Fig. 3.3: fig03_03.cpp
// Define class GradeBook with a member function that takes a parameter;
// Create a GradeBook object and call its displayMessage function.
#include <iostream>
#include <string>
using namespace std;

// 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;
    }
};

// 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
cout << endl; // output a blank line

// 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!
Member Function w/ a Parameter

Parameter Lists

- Additional information needed by a function
- Located in parentheses following the function name
- A function may have any number of parameters
  - Parameters are separated by commas
- The number, order and types of arguments in a function call must match the number, order and types of parameters in the called function’s parameter list
Common Programming Error 4

Placing a semicolon after the right parenthesis enclosing the parameter list of a function definition is a syntax error.
Common Programming Error 5

Defining a function parameter again as a local variable in the function is a compilation error.
To avoid ambiguity, do not use the same names for the arguments passed to a function and the corresponding parameters in the function definition.
Good Programming Practice 2

Choosing meaningful function names and meaningful parameter names makes programs more readable and helps avoid excessive use of comments.
Data Members, *set* Functions and *get* Functions

- **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
Data Members, set Functions and get Functions Cont’d

- Attributes
  - Exist throughout the life of the object
  - Represented as data members
    - Variables in a class definition
  - Each object of class maintains its own copy of attributes
// Fig. 3.5: fig03_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

    private:

    string courseName; // data member
};
// 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

    // Accessing private data outside class definition
}

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!
Good Programming Practice 3

Place a blank line between member-function definitions to enhance program readability.
Data Members, *set* Functions and *get* Functions Cont’d

- **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
  - Data hiding
Data Members, \textit{set} Functions and \textit{get} Functions Cont'd

- Returning a value from a function
  - A function that specifies a return type other than \texttt{void}
    - Must return a value to its calling function
As a rule, 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.)
Common Programming Error 6

An attempt by a function, which is not a member of a particular class (or a friend of that class), to access a private member of that class is a compilation error.
Good Programming Practice 4

Despite the fact that the public and private access specifiers may be repeated and intermixed, list all the public members of a class first in one group and then list all the private members in another group. This focuses the client’s attention on the class’s public interface, rather than on the class’s implementation.
Good Programming Practice 5

If you choose to list the private members first in a class definition, explicitly use the private access specifier despite the fact that private is assumed by default. This improves program clarity.
Functions and classes declared by a class to be friends of that class can access the private members of the class.
Error-Prevention Tip 1

Making the data members of a class private and the member functions of the class public facilitates debugging because problems with data manipulations are localized to either the class’s member functions or the friends of the class.
Common Programming Error 7

Forgetting to return a value from a function that is supposed to return a value is a compilation error.
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*
Data Members, *set* Functions and *get* Functions Cont’d

- Software engineering with *set* and *get* functions Cont’d
  - Using *set* and *get* functions 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

Good Programming Practice 6

Always try to localize the effects of changes to a class’s data members by accessing and manipulating the data members through their get and set functions. Changes to the name of a data member or the data type used to store a data member then affect only the corresponding get and set functions, but not the callers of those functions.
It is important to write programs that are understandable and easy to maintain. Change is the rule rather than the exception. Programmers should anticipate that their code will be modified.
The class designer need not provide set or get functions for each private data item; these capabilities should be provided only when appropriate. If a service is useful to the client code, that service should typically be provided in the class’s public interface.
Initializing Objects w/ Constructors

Constructors

- Functions used to initialize an object’s data when it is created
  - Call made implicitly when object is created
  - Must be defined with the same name as the class
  - Cannot return values
    - Not even void

- Default constructor has no parameters
  - The compiler will provide one when a class does not explicitly include a constructor
    - Compiler’s default constructor only calls constructors of data members that are objects of classes
// Fig. 3.7: fig03_07.cpp
// Instantiating multiple objects of the GradeBook class and using
// the GradeBook constructor to specify the course name
// when each GradeBook object is created.
#include <iostream>
using std::cout;
using std::endl;

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

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

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

    // Constructor has same name as class and no return type

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

// display a welcome message to the GradeBook user
void displayMessage()
{
    // call getCourseName to get the courseName
    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()
{
    // 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() << endl;
    cout << "gradeBook2 created for course: " << gradeBook2.getCourseName() << endl;

    return 0; // indicate successful termination
}

// end main

gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
Error-Prevention Tip 2

Unless no initialization of your class’s data members is necessary (almost never), provide a constructor to ensure that your class’s data members are initialized with meaningful values when each new object of your class is created.
Data members can be initialized in a constructor of the class or their values may be set later after the object is created. However, it is a good software engineering practice to ensure that an object is fully initialized before the client code invokes the object’s member functions. In general, you should not rely on the client code to ensure that an object gets initialized properly.
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
    - Allow compiler to recognize the classes when used elsewhere
  - Generally have .h filename extensions
Placing a Class in a Separate File for Reusability Cont’d

- Driver files
  - Program used to test software (such as classes)
  - Contains a main function so it can be executed

// Fig. 3.9: GradeBook.h

// GradeBook class definition in a separate file from main.
#include <iostream>
using std::cout;
using std::endl;

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

// 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

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

  Class definition is in a header file
// function to get the course name
string getCourseName()
{
    return courseName; // return object's courseName
} // end function getCourseName

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

private:
    string courseName; // course name for this GradeBook
}; // end class GradeBook
// Fig. 3.10: fig03_10.cpp
// Including class GradeBook from file GradeBook.h for use in main.
#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() << endl;
    cout << "gradeBook2 created for course: " << gradeBook2.getCourseName() << endl;

    return 0; // indicate successful termination
} // end main

gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
Placing a Class in a Separate File for Reusability Cont’d

- `#include` preprocessor directive
  - Used to include header files
    - Instructs C++ preprocessor to replace directive with a copy of the contents of the specified file
  - Quotes indicate user-defined header files
    - Preprocessor first looks in current directory
      - If the file is not found, looks in C++ Standard Library directory
Placing a Class in a Separate File for Reusability Cont’d

- `#include` preprocessor directive Cont’d
  - Angle brackets indicate C++ Standard Library
    - Preprocessor looks only in C++ Standard Library directory

- Creating objects
  - Compiler must know size of object
    - C++ objects typically contain only data members
    - Compiler creates one copy of class’s member functions
      - This copy is shared among all the class’s objects
Error-Prevention Tip 3

To ensure that the preprocessor can locate header files correctly, \#include preprocessor directives should place the names of user-defined header files in quotes (e.g., "GradeBook.h") and place the names of C++ Standard Library header files in angle brackets (e.g., <iostream>).
Separating Interface from Implementation

- **Interface**
  - Describes what services a class’s clients can use and how to request those services
    - But does not reveal how the class carries out the services
  - A class definition that lists only member function names, return types and parameter types
    - Function prototypes
  - A class’s interface consists of the class’s public member functions (services)
Separating Interface from Implementation Cont’d

- Separating interface from implementation
  - Client code should not break if the implementation changes, as long as the interface stays the same
  - Define member functions outside the class definition, in a separate source-code file
    - In source-code file for a class
      - Use binary scope resolution operator (::) to “tie” each member function to the class definition

Separating Interface from Implementation Cont’d

- **Separating interface from implementation Cont’d**
  - Define member functions outside the class definition, in a separate source-code file Cont’d
    - Implementation details are hidden
      - Client code does not need to know the implementation

- In the header file for a class
  - Function prototypes describe the class’s public interface

---

// Fig. 3.11: 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

Interface contains data members and member function prototypes
Common Programming Error 8

Forgetting the semicolon at the end of a function prototype is a syntax error.
Although parameter names in function prototypes are optional (they are ignored by the compiler), many programmers use these names for documentation purposes.
Error-Prevention Tip 4

Parameter names in a function prototype (which, again, are ignored by the compiler) can be misleading if wrong or confusing names are used. For this reason, many programmers create function prototypes by copying the first line of the corresponding function definitions (when the source code for the functions is available), then appending a semicolon to the end of each prototype.

When defining a class’s member functions outside that class, omitting the class name and binary scope resolution operator (::) preceding the function names causes compilation errors.
// Fig. 3.12: 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 " << getCourseName()
    << "!" << endl;
}
// end function displayMessage
// Fig. 3.13: fig03_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() << endl;
    cout << "gradeBook2 created for course: " << gradeBook2.getCourseName() << endl;

    return 0; // indicate successful termination
} // end main

gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
Separating Interface from Implementation Cont’d

The Compilation and Linking Process

- Source-code file is compiled to create the class’s object code (source-code file must `#include` header file)
  - Class implementation programmer only needs to provide header file and object code to client
- Client must `#include` header file in their own code
  - So compiler can ensure that the `main` function creates and manipulates objects of the class correctly
Separating Interface from Implementation Cont’d

The Compilation and Linking Process Cont’d

- To create an executable application
  - Object code for client code must be linked with the object code for the class and the object code for any C++ Standard Library object code used in the application
Compilation and Linking Process
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

// Fig. 3.15: GradeBook.h
// GradeBook class definition presents the public interface of
// the class. Member-function definitions appear in GradeBook.cpp.
#include <string> // program uses C++ standard string class
using std::string;

// GradeBook class definition
class GradeBook
{
  public:
    GradeBook( string ); // constructor that initializes a GradeBook object
    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. 3.16: GradeBook.cpp

// Implementations of the GradeBook member-function definitions.
// The 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
    }
}
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).\n" << "Limiting courseName to first 25 characters.\n" << endl;
}

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

// 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;
}
// Fig. 3.17: fig03_17.cpp
// Create and manipulate a GradeBook object; illustrate validation.
#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;
    // initial course name of gradeBook1 is too long
    GradeBook gradeBook1("CS101 Introduction to Programming in C++");
    GradeBook gradeBook2("CS102 C++ Data Structures");
}

Constructor will call set function to perform validity checking.
// display each GradeBook's courseName
18 cout << "gradeBook1's initial course name is: "
19 << gradeBook1.getCourseName()
20 << 
21 << gradeBook2.getCourseName() << endl;

// modify myGradeBook's courseName (with a valid-length string)
24 gradeBook1.setCourseName("CS101 C++ Programming");

// display each GradeBook's courseName
27 cout << "gradeBook1's course name is: "
28 << gradeBook1.getCourseName()
29 << 
30 << gradeBook2.getCourseName() << endl;
31 return 0; // indicate successful termination
32 } // end main

Name "CS101 Introduction to Programming in C++" exceeds maximum length (25). Limiting courseName to first 25 characters.

gradeBook1's initial course name is: CS101 Introduction to Pro
gradeBook2's initial course name is: CS102 C++ Data Structures
gradeBook1's course name is: CS101 C++ Programming
gradeBook2's course name is: CS102 C++ Data Structures
Making data members private and controlling access, especially write access, to those data members through public member functions helps ensure data integrity.
Error-Prevention Tip 5

The benefits of data integrity are not automatic simply because data members are made private—the programmer must provide appropriate validity checking and report the errors.
Member functions that set the values of private data should verify that the intended new values are proper; if they are not, the set functions should place the private data members into an appropriate state.