

# Computer Programming

## C++ Classes and Objects


### 6<sup>th</sup> Lecture

엄현상 (Eom, Hyeonsang)  
School of Computer  
Science and Engineering  
Seoul National University



# 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 and
- 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



# Classes, Objects, Member Functions, & Data Members Cont'd

- Classes: Car example Cont'd
  - 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

```

1 // Fig. 3.1: fig03_01.cpp
2 // Define class GradeBook with a member function displayMessage;
3 // Create a GradeBook object and call its displayMessage function.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 // GradeBook class definition
9 class GradeBook
10 {
11 public:
12     // function that displays a welcome message to the user
13     void displayMessage()
14     {
15         cout << "Welcome to the Grade Book!" << endl;
16     } // end function displayMessage
17 }; // end class GradeBook
18
19 // function main begins program execution
20 int main()
21 {
22     GradeBook myGradeBook; // create a GradeBook object named myGradeBook
23     myGradeBook.displayMessage(); // call object's displayMessage function
24     return 0; // indicate successful termination
25 } // end main

```

Beginning of class definition  
for class **GradeBook**

Beginning of class body

Access specifier **public**; makes  
members available

Member function  
**displayMessage** returns nothing

End of class body

Use dot operator to call  
**GradeBook**'s member function

Welcome to the Grade Book!





# Common Programming Error 1

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



# Class w/ a Member Function Cont'd

## ■ 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

### □ Parentheses 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



# Member Function w/ 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 newline is encountered
- Example
  - `getline( cin, nameOfCourse );`
    - Inputs a line from standard input into string object `nameOfCourse`

```

1 // Fig. 3.3: fig03_03.cpp
2 // Define class GradeBook with a member function that takes a parameter;
3 // Create a GradeBook object and call its displayMessage function.
4 #include <iostream>
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 #include <string> // program uses C++ standard string class
10 using std::string;
11 using std::getline;
12
13 // GradeBook class definition
14 class GradeBook
15 {
16 public:
17     // function that displays a welcome message to the GradeBook user
18     void displayMessage( string courseName )
19     {
20         cout << "Welcome to the grade book for\n" << courseName << "\n"
21             << endl;
22     } // end function displayMessage
23 }; // end class GradeBook
24
25 // function main begins program execution
26 int main()
27 {
28     string nameOfCourse; // string of characters to store the course name
29     GradeBook myGradeBook; // create a GradeBook object named myGradeBook
30

```

Include string class definition

Member function parameter

Use the function parameter as a variable



# GradeBook Program Cont'd

```
31 // prompt for and input course name
32 cout << "Please enter the course name:" << endl;
33 getline( cin, nameOfCourse ); // read a course name with blanks
34 cout << endl; // output a blank line
35
36 // call myGradeBook's displayMessage function
37 // and pass nameOfCourse as an argument
38 myGradeBook.displayMessage( nameOfCourse );
39 return 0; // indicate successful termination
40 } // end main
```

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

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

Passing an argument to  
the member function



# 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.



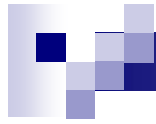
# Common Programming Error 5

Defining a function parameter again as a local variable in the function is a compilation error.




# Good Programming Practice 1

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



```

1 // Fig. 3.5: fig03_05.cpp
2 // Define class GradeBook that contains a courseName data member
3 // and member functions to set and get its value;
4 // Create and manipulate a GradeBook object with these functions.
5 #include <iostream>
6 using std::cout;
7 using std::cin;
8 using std::endl;
9
10 #include <string> // program uses C++ standard string class
11 using std::string;
12 using std::getline;
13
14 // GradeBook class definition
15 class GradeBook
16 {
17 public:
18     // function that sets the course name
19     void setCourseName( string name )
20     {
21         courseName = name; // store the course name in the object
22     } // end function setCourseName
23
24     // function that gets the course name
25     string getCourseName()
26     {
27         return courseName; // return the object's courseName
28     } // end function getCourseName
29

```

*set* function modifies **private** data

*get* function accesses **private** data

```

30 // function that displays a welcome message
31 void displayMessage()
32 {
33     // this statement calls getCourseName to get the
34     // name of the course this GradeBook represents
35     cout << "Welcome to the grade book for\n" << getCourseName() << "!"
36         << endl;
37 } // end function displayMessage
38 private:
39     string courseName; // course name for this GradeBook
40 }; // end class GradeBook
41
42 // function main
43 int main()
44 {
45     string nameOfCourse; // string of characters to store the course name
46     GradeBook myGradeBook; // create a GradeBook object named myGradeBook
47
48     // display initial value of courseName
49     cout << "Initial course name is: " << myGradeBook.getCourseName()
50         << endl;
51

```

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

# GradeBook Program Cont'd

```
52 // prompt for, input and set course name
53 cout << "\nPlease enter the course name:" << endl;
54 getline( cin, nameOfCourse ); // read a course name with blanks
55 myGradeBook.setCourseName( nameOfCourse ); // set the course name
56
57 cout << endl; // outputs a blank line
58 myGradeBook.displayMessage(); // display message with new course name
59 return 0; // indicate success
60 } // end main
```

Modifying **private** data from outside the 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, *set* Functions and *get* Functions Cont'd

- Returning a value from a function
  - A function that specifies a return type other than void
    - Must return a value to its calling function



# Software Engineering Observation 1

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.



# Software Engineering Observation 2

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.



# Data Members, *set* Functions and *get* Functions Cont'd

- 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.





# Software Engineering Observation 3

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.



# Software Engineering Observation 4

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

```

1 // Fig. 3.7: fig03_07.cpp
2 // Instantiating multiple objects of the GradeBook class and using
3 // the GradeBook constructor to specify the course name
4 // when each GradeBook object is created.
5 #include <iostream>
6 using std::cout;
7 using std::endl;
8
9 #include <string> // program uses C++ standard string class
10 using std::string;
11
12 // GradeBook class definition
13 class GradeBook
14 {
15 public:
16     // constructor initializes courseName with string supplied as argument
17     GradeBook( string name )
18     {
19         setCourseName( name ); // call set function to initialize courseName
20     } // end GradeBook constructor
21
22     // function to set the course name
23     void setCourseName( string name )
24     {
25         courseName = name; // store the course name in the object
26     } // end function setCourseName
27

```

Constructor has same name as class and no return type

Initialize data member



# GradeBook Program Cont'd

```
28 // function to get the course name
29 string getCourseName()
30 {
31     return courseName; // return object's courseName
32 } // end function getCourseName
33
34 // display a welcome message to the GradeBook user
35 void displayMessage()
36 {
37     // call getCourseName to get the courseName
38     cout << "Welcome to the grade book for\n" << getCourseName()
39         << "!" << endl;
40 } // end function displayMessage
41 private:
42     string courseName; // course name for this GradeBook
43 }; // end class GradeBook
44
```

# GradeBook Program Cont'd

```
45 // function main begins program execution
46 int main()
47 {
48     // create two GradeBook objects
49     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
50     GradeBook gradeBook2( "CS102 Data Structures in C++" );
51
52     // display initial value of courseName for each GradeBook
53     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
54         << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
55         << endl ;
56     return 0; // indicate successful termination
57 } // end main
```

Creating objects implicitly calls the constructor

```
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.



# Software Engineering Observation 5

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

```
1 // Fig. 3.9: GradeBook.h
2 // GradeBook class definition in a separate file from main.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <string> // class GradeBook uses C++ standard string class
8 using std::string;
9
10 // GradeBook class definition
11 class GradeBook
12 {
13 public:
14     // constructor initializes courseName with string supplied as argument
15     GradeBook( string name )
16     {
17         setCourseName( name ); // call set function to initialize courseName
18     } // end GradeBook constructor
19
20     // function to set the course name
21     void setCourseName( string name )
22     {
23         courseName = name; // store the course name in the object
24     } // end function setCourseName
25
```

Class definition is in a header file



# GradeBook Program Cont'd

```
26 // function to get the course name
27 string getCourseName()
28 {
29     return courseName; // return object's courseName
30 } // end function getCourseName
31
32 // display a welcome message to the GradeBook user
33 void displayMessage()
34 {
35     // call getCourseName to get the courseName
36     cout << "Welcome to the grade book for\n" << getCourseName()
37         << "!" << endl;
38 } // end function displayMessage
39 private:
40     string courseName; // course name for this GradeBook
41 }; // end class GradeBook
```

```

1 // Fig. 3.10: fig03_10.cpp
2 // Including class GradeBook from file GradeBook.h for use in main.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "GradeBook.h" // include definition of class GradeBook
8
9 // function main begins program execution
10 int main()
11 {
12     // create two GradeBook objects
13     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
14     GradeBook gradeBook2( "CS102 Data Structures in C++" );
15
16     // display initial value of courseName for each GradeBook
17     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
18         << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
19         << endl;
20     return 0; // indicate successful termination
21 } // end main

```

Including the header file causes the class definition to be copied into the file

```

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

# GradeBook Program Cont'd

```
1 // Fig. 3.11: GradeBook.h
2 // GradeBook class definition. This file presents GradeBook's public
3 // interface without revealing the implementations of GradeBook's member
4 // functions, which are defined in GradeBook.cpp.
5 #include <string> // class GradeBook uses C++ standard string class
6 using std::string;
7
8 // GradeBook class definition
9 class GradeBook
10 {
11 public:
12     GradeBook( string ); // constructor that initializes courseName
13     void setCourseName( string ); // function that sets the course name
14     string getCourseName(); // function that gets the course name
15     void displayMessage(); // function that displays a welcome message
16 private:
17     string courseName; // course name for this GradeBook
18 }; // 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.



# Good Programming Practice 7

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.



## Common Programming Error 9

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.



# GradeBook Program Cont'd

```
1 // Fig. 3.12: GradeBook.cpp
2 // GradeBook member-function definitions. This file contains
3 // implementations of the member functions prototyped in GradeBook.h.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13     setCourseName( name ); // call set function to initialize courseName
14 } // end GradeBook constructor
15
16 // function to set the course name
17 void GradeBook::setCourseName( string name )
18 {
19     courseName = name; // store the course name in the object
20 } // end function setCourseName
21
```

GradeBook implementation is placed in a separate source-code file

Include the header file to access the class name **GradeBook**

Binary scope resolution operator “ties” a function to its class



# GradeBook Program Cont'd

```
22 // function to get the course name
23 string GradeBook::getCourseName()
24 {
25     return courseName; // return object's courseName
26 } // end function getCourseName
27
28 // display a welcome message to the GradeBook user
29 void GradeBook::displayMessage()
30 {
31     // call getCourseName to get the courseName
32     cout << "Welcome to the grade book for\n" << getCourseName()
33         << "!" << endl;
34 } // end function displayMessage
```

```


1 // Fig. 3.13: fig03_13.cpp
2 // GradeBook class demonstration after separating
3 // its interface from its implementation.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "GradeBook.h" // include definition of class GradeBook
9
10 // function main begins program execution
11 int main()
12 {
13     // create two GradeBook objects
14     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
15     GradeBook gradeBook2( "CS102 Data Structures in C++" );
16
17     // display initial value of courseName for each GradeBook
18     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
19         << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
20         << endl;
21     return 0; // indicate successful termination
22 } // 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

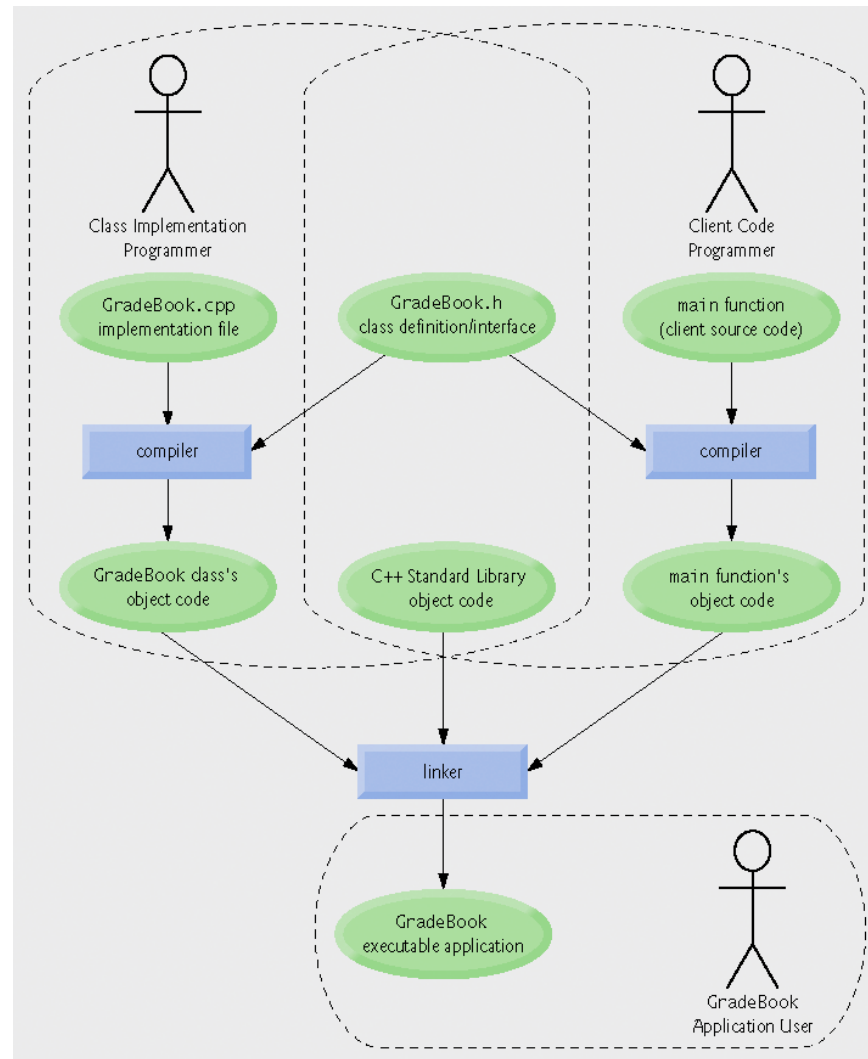
- So compiler can ensure that the main function creates and manipulates objects of the class correctly



# 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*



# GradeBook Program Cont'd

```
1 // Fig. 3.15: GradeBook.h
2 // GradeBook class definition presents the public interface of
3 // the class. Member-function definitions appear in GradeBook.cpp.
4 #include <string> // program uses C++ standard string class
5 using std::string;
6
7 // GradeBook class definition
8 class GradeBook
9 {
10 public:
11     GradeBook( string ); // constructor that initializes a GradeBook object
12     void setCourseName( string ); // function that sets the course name
13     string getCourseName(); // function that gets the course name
14     void displayMessage(); // function that displays a welcome message
15 private:
16     string courseName; // course name for this GradeBook
17 }; // end class GradeBook
```



# GradeBook Program Cont'd

```
1 // Fig. 3.16: GradeBook.cpp
2 // Implementations of the GradeBook member-function definitions.
3 // The setCourseName function performs validation.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13     setCourseName( name ); // validate and store courseName
14 } // end GradeBook constructor
15
16 // function that sets the course name;
17 // ensures that the course name has at most 25 characters
18 void GradeBook::setCourseName( string name )
19 {
20     if ( name.length() <= 25 ) // if name has 25 or fewer characters
21         courseName = name; // store the course name in the object
22
```

Constructor calls *set* function to perform validity checking

*set* functions perform validity checking to keep **courseName** in a consistent state



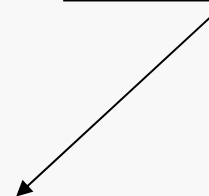
# GradeBook Program Cont'd

```
23  if ( name.length() > 25 ) // if name has more than 25 characters
24  {
25      // set courseName to first 25 characters of parameter name
26      courseName = name.substr( 0, 25 ); // start at 0, length of 25
27
28      cout << "Name \" " << name << "\" exceeds maximum length (25).\n"
29          << "Limiting courseName to first 25 characters.\n" << endl ;
30  } // end if
31 } // end function setCourseName
32
33 // function to get the course name
34 string GradeBook::getCourseName()
35 {
36     return courseName; // return object's courseName
37 } // end function getCourseName
38
39 // display a welcome message to the GradeBook user
40 void GradeBook::displayMessage()
41 {
42     // call getCourseName to get the courseName
43     cout << "Welcome to the grade book for\n" << getCourseName()
44         << "!" << endl ;
45 } // end function displayMessage
```

# GradeBook Program Cont'd

```
1 // Fig. 3.17: fig03_17.cpp
2 // Create and manipulate a GradeBook object; illustrate validation.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "GradeBook.h" // include definition of class GradeBook
8
9 // function main begins program execution
10 int main()
11 {
12     // create two GradeBook objects;
13     // initial course name of gradeBook1 is too long
14     GradeBook gradeBook1( "CS101 Introduction to Programming in C++" );
15     GradeBook gradeBook2( "CS102 C++ Data Structures" );
16 }
```

Constructor will call *set* function to perform validity checking



# GradeBook Program Cont'd

```
17 // display each GradeBook's courseName
18 cout << "gradeBook1's initial course name is: "
19     << gradeBook1.getCourseName()
20     << "\ngradeBook2's initial course name is: "
21     << gradeBook2.getCourseName() << endl;
22
23 // modify myGradeBook's courseName (with a valid-length string)
24 gradeBook1.setCourseName("CS101 C++ Programming");
25
26 // display each GradeBook's courseName
27 cout << "\ngradeBook1's course name is: "
28     << gradeBook1.getCourseName()
29     << "\ngradeBook2's course name is: "
30     << gradeBook2.getCourseName() << endl;
31 return 0; // indicate successful termination
32 } // end main
```

Call *set* function to perform validity checking

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



# Software Engineering Observation 6

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.



# Software Engineering Observation 7

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.