Good Programming Practice 3.2

Although parameter names in function prototypes are optional (they're ignored by the compiler), many programmers use these names for documentation purposes.

3.7 Separating Interface from Implementation (cont.)

- Source-code file `GradeBook.cpp` (Fig. 3.12) *defines* class `GradeBook`’s member functions, which were declared in lines 11–14 of Fig. 3.11.
- Each member-function name (lines 9, 16, 22 and 28) is preceded by the class name and `::`, which is known as the scope resolution operator.
- This “ties” each member function to the (now separate) `GradeBook` class definition (Fig. 3.11), which declares the class’s member functions and data members.

```cpp
// Fig. 3.12: GradeBook.cpp
// GradeBook member-function definitions. This file contains
// implementations of the member functions prototyped in GradeBook.h.
#include <iostream>
#include "GradeBook.h" // include definition of class GradeBook
using namespace std;

// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name )
   : courseName( name ) // member initializer to initialize courseName
{
   // empty body
} // 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

```

Fig. 3.12 | GradeBook member-function definitions represent the implementation of class GradeBook. (Part 1 of 2.)

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

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

Fig. 3.12 | GradeBook member-function definitions represent the implementation of class GradeBook. (Part 2 of 2.)

Common Programming Error 3.3

When defining a class's member functions outside that class, omitting the class name and scope resolution operator (`::`) preceding the function names causes errors.

3.7 Separating Interface from Implementation (cont.)

- To indicate that the member functions in `GradeBook.cpp` are part of class `GradeBook`, we must first include the `GradeBook.h` header file (line 5 of Fig. 3.12).
- This allows us to access the class name `GradeBook` in the `GradeBook.cpp` file.
- When compiling `GradeBook.cpp`, the compiler uses the information in `GradeBook.h` to ensure that
  - the first line of each member function matches its prototype in the `GradeBook.h` file, and that
  - each member function knows about the class's data members and other member functions

```cpp
// Fig. 3.13: fig03_13.cpp
// GradeBook class demonstration after separating
// its interface from its implementation.
#include <iostream>
#include "GradeBook.h" // include definition of class GradeBook
using namespace std;

// 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;
} // end main
```

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

Fig. 3.13 | GradeBook class demonstration after separating its interface from its implementation.

3.7 Separating Interface from Implementation (cont.)

- Before executing this program, the source-code files in Fig. 3.12 and Fig. 3.13 must both be compiled, then linked together—that is, the member-function calls in the client code need to be tied to the implementations of the class's member functions—a job performed by the linker.
- The diagram in Fig. 3.14 shows the compilation and linking process that results in an executable `GradeBook` application that can be used by instructors

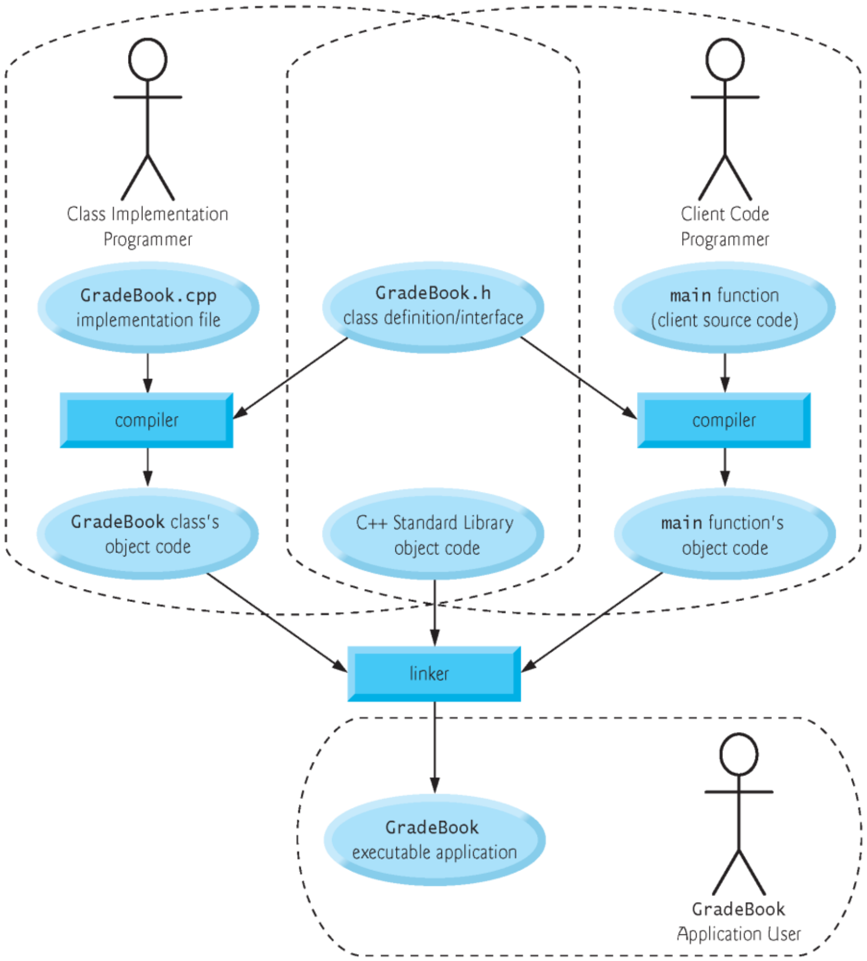
Class Implementation Programmer
Client Code Programmer
GradeBook.cpp implementation file
GradeBook.h class definition/interface
main function (client source code)
compiler
compiler
GradeBook class's object code
C++ Standard Library object code
main function's object code
linker
GradeBook executable application
GradeBook Application User

Fig. 3.14 | Compilation and linking process that produces an executable application.

3.8 Validating Data with *set Functions*

- The program of Figs. 3.15–3.17 enhances class `GradeBook`'s member function `setCourseName` to perform validation (also known as validity checking).
- Since the interface of the clas remains unchanged, clients of this class need not be changed when the definition of member function `setCourseName` is modified.
- This enables clients to take advantage of the improved `GradeBook` class simply by

```cpp
// 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

// GradeBook class definition
class GradeBook
{
public:
   explicit GradeBook( std::string ); // constructor initialize courseName
   void setCourseName( std::string ); // sets the course name
   std::string getCourseName() const; // gets the course name
   void displayMessage() const; // displays a welcome message
private:
   std::string courseName; // course name for this GradeBook
}; // end class GradeBook
```

Fig. 3.15 | GradeBook class definition presents the `public` interface of the class.

3.8 Validating Data with *set Functions (cont.)*

- The C++ Standard Library’s `string` class defines a member function `length` that returns the number of characters in a `string` object.
- A consistent state is a state in which the object’s data member contains a valid value.
- Class `string` provides member function `substr` (short for “substring”) that returns a new `string` object created by copying part of an existing `string` object.
  - The first argument specifies the starting position in the original `string` from which characters are copied.
  - The second argument specifies the number of characters to copy.

3.10 Validating Data with *set Functions (cont.)*

- Figure 3.17 demonstrates the modified version of class `GradeBook` (Figs. 3.15–3.16) featuring validation.
- In previous versions of the class, the benefit of calling `setCourseName` in the constructor was not evident.
- Now, however, *the constructor takes advantage of the validation* provided by `setCourseName`.
- The constructor simply calls

```cpp
// Fig. 3.16: GradeBook.cpp
// Implementations of the GradeBook member-function definitions.
// The setCourseName function performs validation.
#include <iostream>
#include "GradeBook.h" // include definition of class GradeBook
using namespace std;

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

```

Fig. 3.16 | Member-function definitions for class `GradeBook` with a *set* function that validates the length of data member `courseName`. (Part 1 of 3.)

```cpp
// function that sets the course name;
// ensures that the course name has at most 25 characters
void GradeBook::setCourseName( string name )
{
   if ( name.size() <= 25 ) // if name has 25 or fewer characters
      courseName = name; // store the course name in the object

   if ( name.size() > 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

      cerr << "Name \"" << name << "\" exceeds maximum length (25).\n"
         << "Limiting courseName to first 25 characters.\n" << endl;
   } // end if
} // end function setCourseName

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

Fig. 3.16 | Member-function definitions for class GradeBook with a *set* function that validates the length of data member courseName. (Part 2 of 3.)