UEE1303
Objective-Oriented Programming

C++_Lecture 03:

Classes and Objects

C: How to Program 8th ed.

Agenda

- Procedural vs. object-oriented programming
- C++ structure
- Classes
  - Accessing class members
  - Member functions
- Constructor and destructor function
  - Constructors
  - Destructors

Procedural vs. Object-oriented

- C++ supports procedural programming which
  - solves a variety of engineering problems
  - decreasingly efficient for large and complex program development
- C++ also supports object-oriented programming which
  - is more natural as the logic applied to real-life problem
  - programmers who have used procedural methodology for years often have difficulties adopting this logic

Procedural Programming's Problem

- Procedural programming paradigm focus on program's functionality
  - How to represent data is not concerned
  - `main` + `func_1` + `func_2` + … + `func_N`
- For large and complex programs, procedural programming faces the difficulties of
  - maintaining and modifying the program
  - debugging and following its logic
  - too many disorganized, overloaded detail
  - creation of inadvertent logic errors

Sample Problem from Structure

```
struct SScore {
    char name[20];
    int  subj[3];
};
double computeAverage(SScore one) {
    return (one.subj[0]+one.subj[1]+
            one.subj[2])/3;
}
```

- Potential problems:
  - the number of subjects is changed to 4
  - `computeAverage()` may change the user name unintentionally

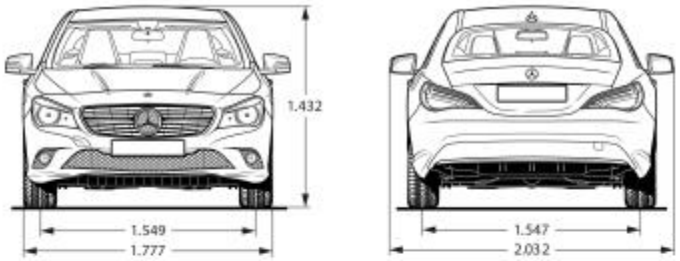
OOP Paradigm

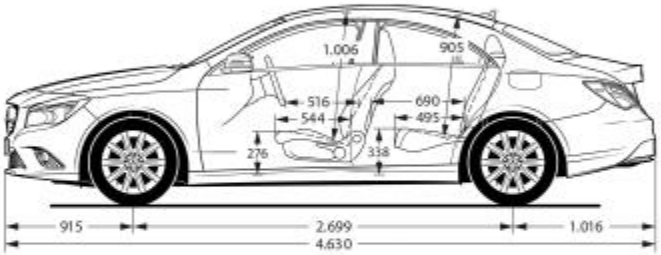
- OOP overcomes the above problems by
  - using a collection of objects that communicate with each other through their interface functions
  - focusing on both data and operations
- Three important concepts in OOP
  - Encapsulation binds data and functions into one capsule (object)
  - Inheritance enables new codes to be derived from existing codes
  - Polymorphism uses the same functions on different types of objects

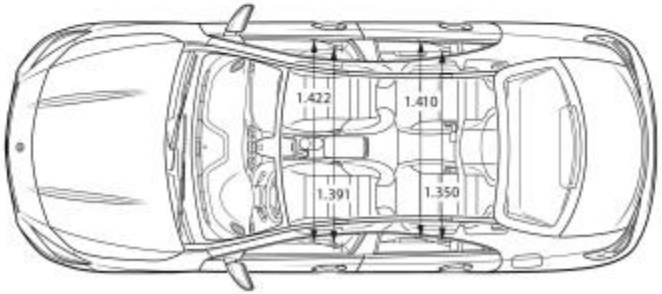
Concept of Class

- Class is a foundation of OOP
  - expands structure by binding together data and functions
  - variable in class types are objects
- A object has its own unique identify, attributes and behaviors
  - attribute ⇨ data fields (data member)
  - behaviors ⇨ a lot of functions (member function)
- Class is the abstraction of objects ⇔ a object is an instance of class
  - Class is a abstract ⇨ takes no memory
  - Object is concrete ⇨ takes memory space

Concept of Class (cont.)







Class Declaration

- Defined similar to structures

```
class class_name                          Class Head
{
public:        Member access modifier     Class Body
     //public variables and functions
protected:
     //protected variables and functions
private:
     //private variables and functions
};
```

  - Class name must be a legal identifier
  - Class body includes many data members (variables) and member functions (methods)

Class Declaration (cont.)

- Member access modifiers can appear in an arbitrary order or multiple times
  - `public`: members can be accessed by members of its class or those of any other class or any non-member function
  - `private`: members serve only as utility functions for others of the same class
  - `protected`: used only with inheritance; members can be accessed by other members of its class or the derived class
- The default modifier for class is `private` ⇔ the default for struct is `public`

Class Declaration (cont.)

- Example

```
class Time  // name of new class type
{
public:             // member access modifier!
    // member function!
    void setTime(int, int, int);
    int hour;
    int minute;
    int second;
};
```

  - Notice only member function's prototype
    - Function's implementation is elsewhere

- Declaration

```
Time t1, t2;
```

Public and Private Example

```
class Time
{
public:
      void setTime(int, int, int);
      void printUniversal();
      void printStandard();
private:
      int hour;
      int minute;
      int second;
};
Time t1;
```

- `cin >> t1.hour;   // NOT ALLOWED!`
- `cout << t1.minute;   // NOT ALLOWED!`
- Must instead call public operations:
  - `t1.printUniversal();`
  - `t1.printStandard();`

Two ways to Define Classes

- First declare class, then define objects ⇨ the most common

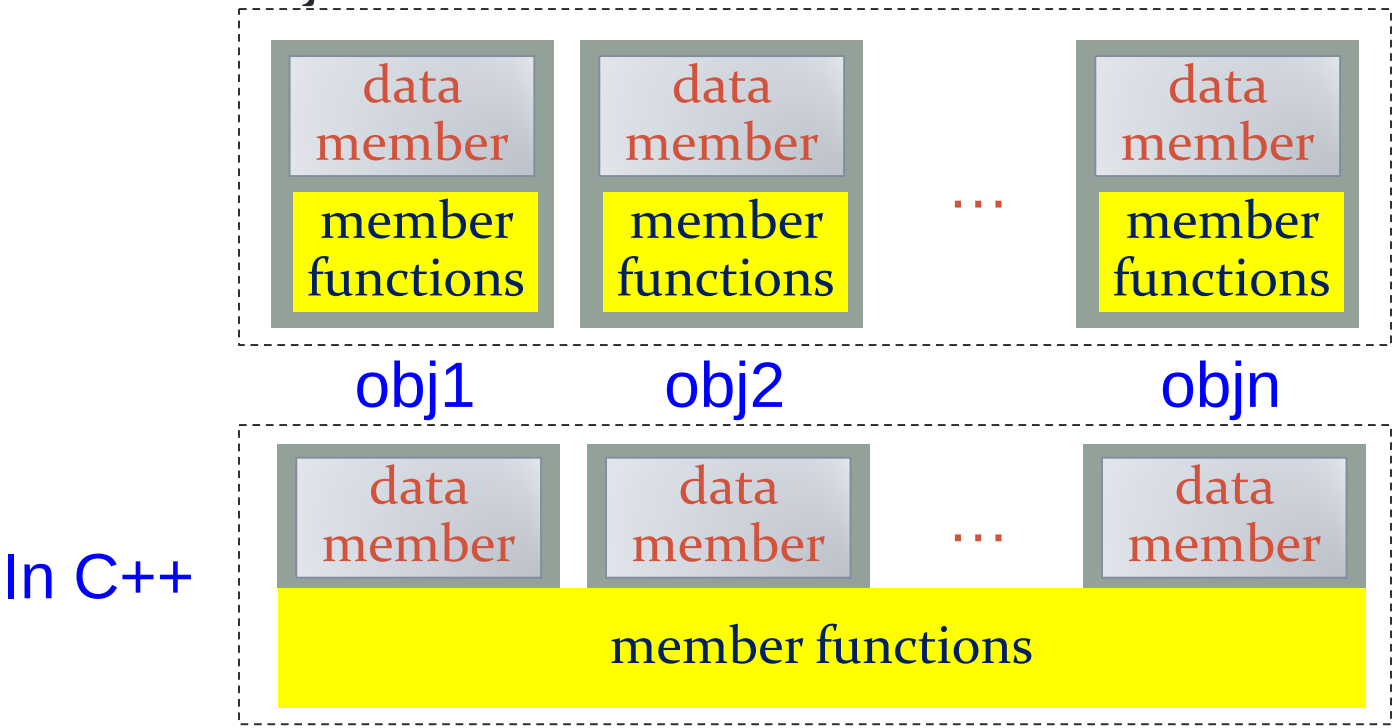
```
class class_name
{
    ...//implement data and functions
};
class_name obj1, obj2, ..., objn;
```

- Declare class and define objects right away

```
class class_name
{
    ...//implement data and functions
} obj1, obj2, ..., objn;
```

Memory Allocation for Objects

- Need to allocate memory for data members and member functions of every object
  - Objects of the same class use the same functions ⇨ memory waste



Access Members in Objects

- Members in objects can be accessed by

  (1) object name and dot (.) operator

```
Time t1;
t1.setTime(13, 4, 3);
```

  (2) a pointer to the object and arrow (->) operator

```
Time *pTime = &t1;
pTime->setTime(13, 4, 3);
```

  (3) a reference to the object and dot (.) operator

```
Time &rTime = t1;
rTime.setTime(13, 4, 3);
```

Functions in Class

- Member functions are one kind of functions
  - Class without member functions = struct
  - Belongs to class need to consider accessibility (access modifier)
- Member functions can be defined into
  - Functions defined in the class body: default as inline functions that are allowed to be included in the header files
  - Function declared in the class body: the typical case; function definitions are written outside the class

Member Functions

- Example for member functions in class body

```
class Time
{
public:
    void setTime(int h, int m, int s) {
        hour = ( h>=0 && h<24 )? h : 0;
        minute = ( m>=0 && m<60 )? m : 0;
        second = ( s>=0 && s<60 )? s : 0;
    }
    ...
private:
    int hour;
    int minute;
    int second;
};
```

Member Functions (cont.)

- Example for member functions outside class body

```
class Time
{
public:
    void setTime(int, int, int);
    ...
private:
    int hour;
    int minute;
    int second;
};
void Time::setTime(int h, int m, int s) {
        hour = ( h>=0 && h<24 )? h : 0;
        minute = ( m>=0 && m<60 )? m : 0;
        second = ( s>=0 && s<60 )? s : 0;
}
```

Member Functions (cont.)

- Separating declaration from implementation is due to (1) information hiding and (2) intellectual property protection
- You are free to change the implementation but the client program needs not to change as long as the declaration is the same

```
double CScore::computeAverage()
{
    return ((1*subj[0]+2*subj[1]+
    3*subj[2])/3.0); //weighted version
}
```

- As a software vendor, only provide the customer with the header file and class object code without revealing the source

Set and Get Functions

- The private data field cannot be accessed outside the class
  - to make them readable, provide a **get** function to return the field's value ⇨ accessor function

```
returnType getPropertyName()
```

  - to make them updatable, provide a **set** function to set a new value in the field ⇨ mutator function

```
void setPropertyName(datatype value)
```

Example of Set Function

```
// Set (mutator) function
void Time::setTime(int h, int m, int s)
{
    setHour(h); // set private field hour
    setMinute(m); // set private field minute
    setSecond(s); // set private field second
}
void Time::setHour(int h){
    hour = ( h >= 0 && h < 24 )? h : 0;
}
void Time::setMinute(int m){
    minute = ( m >= 0 && m < 60 )? m : 0;
}
void Time::setSecond(int s){
    second = ( s >= 0 && s < 60 )? s : 0;
}
```

Example of Get Function

```
// Get (accessor)function
int Time::getHour()
{
    return hour;
}
int Time::getMinute()
{
    return minute;
}
int Time::getSecond()
{
    return second;
}
```

Time Class

```
// Fig. 9.01: fig09_01.cpp
// modified from C++:How to Programming
#include <iostream>
#include <iomanip>
using namespace std;
// Time class definition
class Time
{
public:
    Time(); // constructor
    void setTime(int, int, int);
    void printUniversal();
    void printStandard();
private:
    int hour;   // 0 – 23 (24-hour clock format)
    int minute; // 0 – 59
    int second; // 0 – 59
}; // end class Time
```

Time Class (cont.)

```
Time::Time() {
    hour = minute = second = 0;
}
void Time::setTime(int h, int m, int s) {
    hour = ( h >= 0 && h < 24 )? h : 0;
    minute = ( m >= 0 && m < 60 )? m : 0;
    second = ( s >= 0 && s < 60 )? s : 0;
}
// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal() {
    cout << setfill('0') << setw(2) << hour << ":"
         << setw(2) << minute << ":" << setw(2) << second;
}
// print Time in stardard-time format (HH:MM:SS AM or PM)
void Time::printStandard()
{
    cout << ( (hour == 0 || hour == 12) ? 12 : hour % 12 )
          << ":" << setfill( '0' ) << setw( 2 ) << minute
          << ":" << setw( 2 ) << second
          << (hour < 12 ? " AM" : " PM");
}
```

Time Class (cont.)

```
int main()
{
   Time t; // create Time object

   // output Time object t's initial values
   cout << "The initial universal time is "
   t.printUniversal(); // 00:00:00
   cout << "\nThe initial standard time is ";
   t.printStandard(); // 12:00:00 AM

   t.setTime(13, 27, 6); // change time
   // output Time object t's new values
   cout << "\n\nThe universal time after setTime is ";
   t.printUniversal(); // 13:27:06
   cout << "\nThe standard time after setTime is ";
   t.printStandard(); // 1:27:06 PM
   return 0;
}
```

Separating Interface from Implementation

- The interface of a class describe
  - what services a class's clients can use
  - how to request those services

- It's better software engineering to define member functions outside the class definition, so that their implementation details can be hidden from the client code.
  - Header file (.h)
  - Source coed file (.cpp)

Time.h

```
// Fig. 9.3: Time.h
// Declaration of class Time
// Member functions are defined in Time.cpp
#ifndef TIME_H
#define TIME_H
class Time
{
public:
    Time(); // constructor
    void setTime(int, int, int);
    void printUniversal();
    void printStandard();
private:
    int hour;   // 0 – 23 (24-hour clock format)
    int minute; // 0 – 59
    int second; // 0 – 59
}; // end class Time
#endif
```

Time.cpp (1/2)

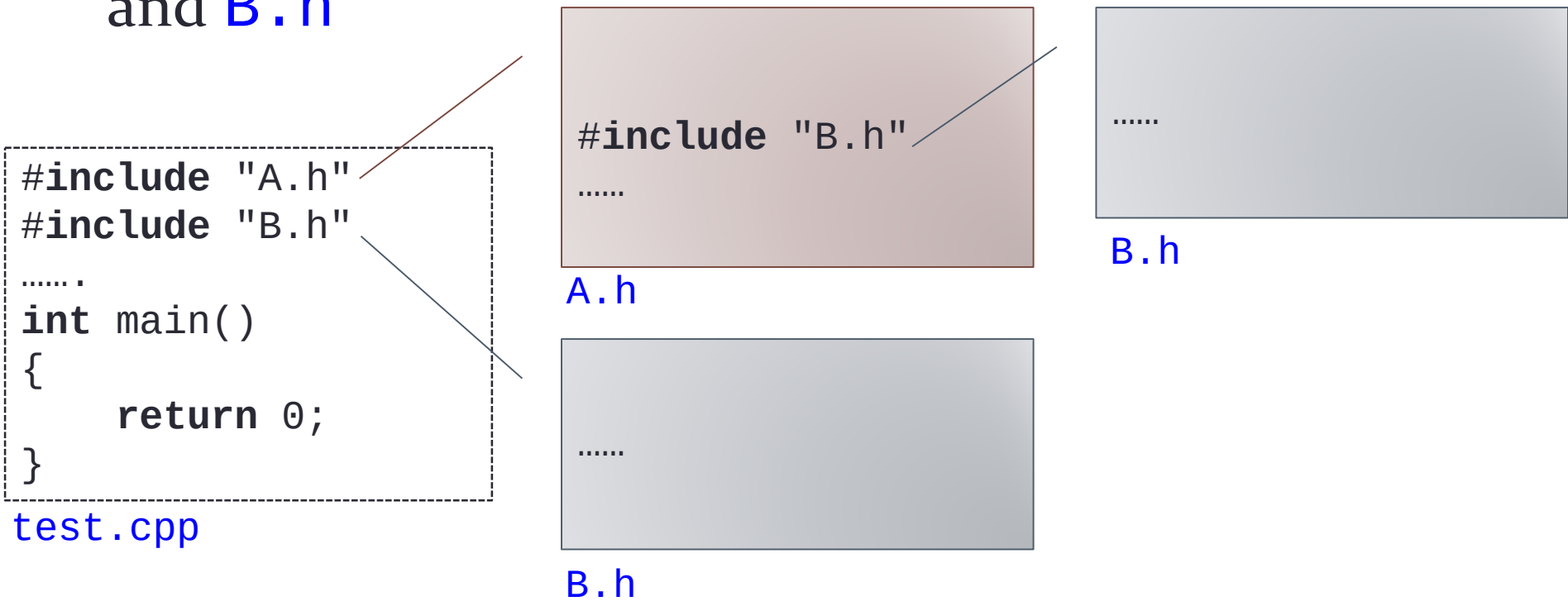
```
// Fig. 9.4: Time.cpp
// Member-function definitions for class Time
#include <iostream>
#include <iomanip>
#include "Time.h"
using namespace std;
// Time constructor initializes each data member to zero;
// ensures that Time object start in a consistent state
Time::Time() {
    hour = minute = second = 0;
}
void Time::setTime(int h, int m, int s) {
    hour = ( h >= 0 && h < 24 )? h : 0;
    minute = ( m >= 0 && m < 60 )? m : 0;
    second = ( s >= 0 && s < 60 )? s : 0;
}
```

Time.cpp (2/2)

```
// set new Time value using universal time; ensure that
// the data remains consistent by setting invalid values
// to zero
void Time::setTime(int h, int m, int s) {
    hour = ( h >= 0 && h < 24 )? h : 0;
    minute = ( m >= 0 && m < 60 )? m : 0;
    second = ( s >= 0 && s < 60 )? s : 0;
}
// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal() {
    cout << setfill('0') << setw(2) << hour << ":"
         << setw(2) << minute << ":" << setw(2) << second;
}
// print Time in stardard-time format (HH:MM:SS AM or PM)
void Time::printStandard()
{
    cout << ( (hour == 0 || hour == 12) ? 12 : hour % 12 )
          << ":" << setfill( '0' ) << setw( 2 ) << minute
          << ":" << setw( 2 ) << second
          << (hour < 12 ? " AM" : " PM");
}
```

Preventing Multiple Declarations

- A common compiling error is to include the same header files multiple times in a program
- Ex: `A.h` includes `B.h` and `test.cpp` includes `A.h` and `B.h`



Preprocessor Wrapper

- When we build larger programs, other definitions and declarations will also be placed in header files.

- The preceding preprocessor wrapper prevents the code between `#ifndef` (which means "if not defined") and `#endif` from being included if the name `TIME_H` has been defined.
  - If the header has not been included previously in a file, the name `TIME_H` is defined by the `#define` directive and the header file statements are included.
  - If the header has been included previously, `TIME_H` is defined already and the header file is not included again

Preprocessor Wrapper (cont.)

- Preprocessor wrapper (Preprocessor directives) solve the issue.

```
#include "A.h"
#include "B.h"
…….
int main()
{
    return 0;
}
```

test.cpp

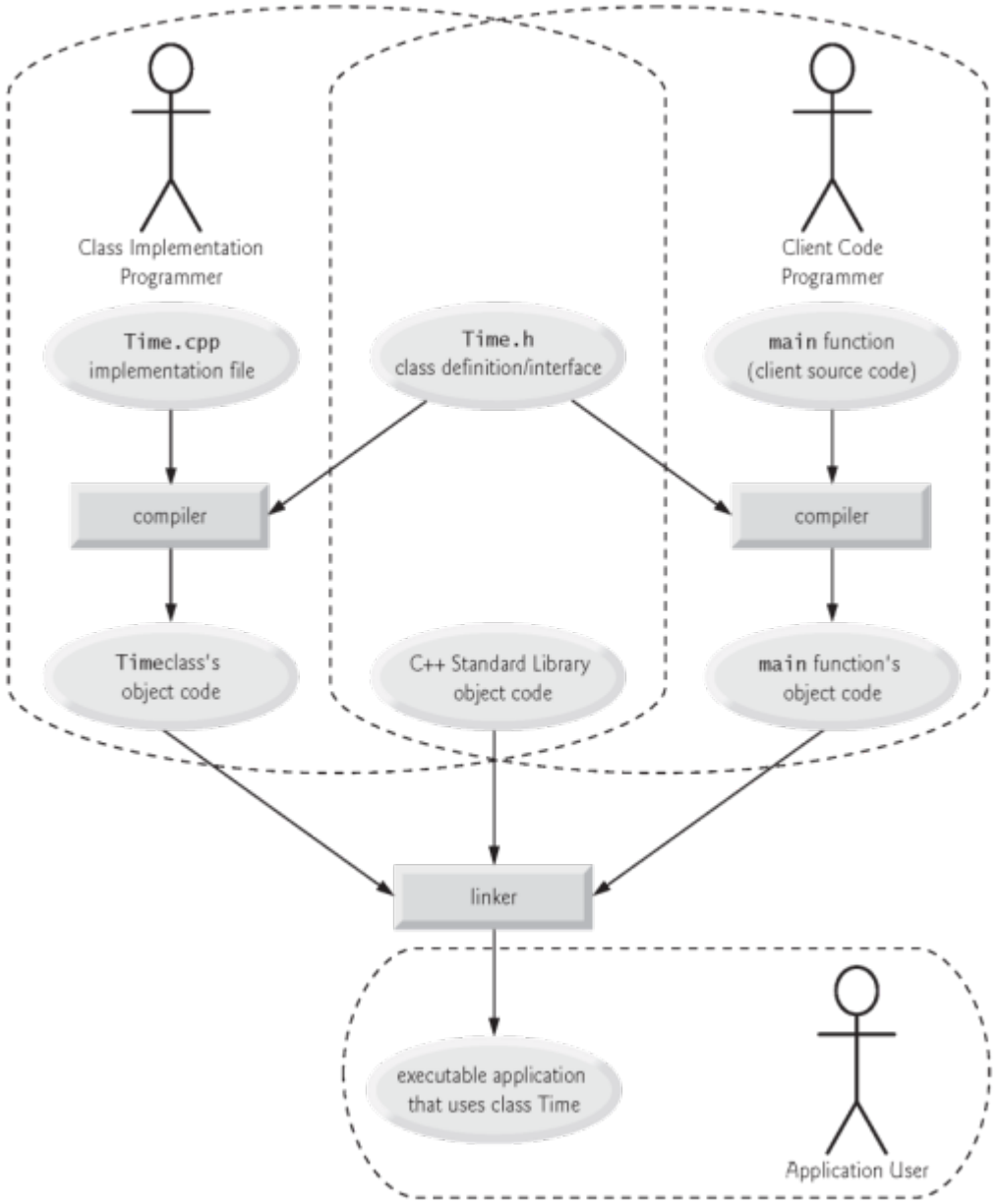
```
#ifndef A_H
#define A_H
#include "B.h"
……
#endif
```

A.h

```
#ifndef B_H
#define B_H
……
#endif
```

```
#ifndef B_H
#define B_H
……
#endif
```

B.h


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

Constructor and Destructors

- Many errors starts from incorrect initialization or clearance of variables in C++
  - two special member functions: constructors and destructors
- Constructors aims at assigning values for data members when creating objects ⇨ object initialization
- Destructors aims at freeing memory space for data members when destroying objects ⇨ object cleaning

Constructor Functions

- A special kind of member function
  - Automatically called when object declared

```
class Time
{
public:
    Time(); // constructor
    void setTime(int, int, int);
    void printUniversal();
    void printStandard();
private:
    int hour;   // 0 – 23 (24-hour clock format)
    int minute; // 0 – 59
    int second; // 0 – 59
}; // end class Time
```

Constructor Functions (cont.)

- A constructor function has the name as the class itself with or without parameters
  - can be overloaded ⇨ multiple functions with the same names

```
Time();
Time(int h);
Time(int h, int m);
Time(int h, int m, int s);
```

  - can have default parameter values

```
Time(int = 0; int = 0, int = 0);
```

Constructor Code

- Constructor definition is like all other member functions but
  - have the same name as the class
  - cannot specify the return type, even void
  - must be public (or can be protected for derived classes)

```
Time::Time(int h, int m, int s)
{
    hour = h;
    minute = m;
    second = s;
}
```

Alternative Definition (Preferable)

```
Time::Time(int h, int m, int s)
{
    hour = h;
    minute = m;
    second = s;
}
```

- It is equivalent to (preferable version)

```
Time::Time(int h, int m, int s)
                :hour(h), minute(m), second(s)
{
 // body intentionally empty
}
```

Initialization Section

Example of Constructor Versions

```
Time::Time(int h, int m, int s)
                :hour(h), minute(m), second(s)
{
    testData();
}
```

```
Time::Time(int h)
                :hour(h), minute(0), second(0)
{
    testData();
}
```

```
Time::Time():hour(0), minute(0), second(0)
{
}
```

Example of Constructor Versions (cont.)

```
void Time::testData() {
    if (hour < 0 || hour > 24)
        hour = 0;
    if (minute < 0 || minute > 60)
        minute = 0;
    if (second < 0 || second > 60)
        second = 0;
}
```

Constructor Call

- May call different versions of constructors

```
Time t1(12, 2, 12);    // 12:02:12
Time t2(3);            // 03:00:00
Time t3;               // 00:00:00
t1 = Time(14, 15, 13);// 14:15:13
```

  - imply overloaded functions
- In ACTION:

```
Time t1(12, 2, 12);
```

  - Constructor called at object's declaration
  - Now to "re-initialize"

```
t1 = Time(14, 15, 13);
```

    - explicit constructor call
    - return new "anonymous object"
    - assigned back to current object

Default Constructor

- If no constructor is defined, the compile implicitly generate a default constructor without any parameter

```
Time::Time() {}
```

- If any constructor is defined, need to specify a default constructor explicitly

```
//error if no default constructor
Time t3;
```

Class with Constructors Example

Display 7.1 Class with Constructors

```
1   #include <iostream>
2   #include <cstdlib> //for exit
3   using namespace std;

4   class DayOfYear
5   {
6   public:
7       DayOfYear(int monthValue, int dayValue);
8       //Initializes the month and day to arguments.

9       DayOfYear(int monthValue);
10      //Initializes the date to the first of the given month.

11      DayOfYear( );
12      //Initializes the date to January 1.

13      void input( );
14      void output( );
15      int getMonthNumber( );
16      //Returns 1 for January, 2 for February, etc.
```

This definition of `DayOfYear` *is an improved version of the class* `DayOfYear` *given in Display 6.4.*

default constructor

Class with Constructors Example (cont.)

```
17      int getDay( );
18  private:
19      int month;
20      int day;
21      void testDate( );
22  };

23  int main( )
24  {
25      DayOfYear date1(2, 21), date2(5), date3;
26      cout << "Initialized dates:\n";
27      date1.output( ); cout << endl;
28      date2.output( ); cout << endl;
29      date3.output( ); cout << endl;

30      date1 = DayOfYear(10, 31);
31      cout << "date1 reset to the following:\n";
32      date1.output( ); cout << endl;
33      return 0;
34  }
35
36  DayOfYear::DayOfYear(int monthValue, int dayValue)
37                               : month(monthValue), day(dayValue)
38  {
39      testDate( );
40  }
```

This causes a call to the default constructor. Notice that there are no parentheses.

an explicit call to the constructor `DayOfYear::DayOfYear`

Class with Constructors Example (cont.)

Display 7.1 Class with Constructors

```
41  DayOfYear::DayOfYear(int monthValue) : month(monthValue), day(1)
42  {
43      testDate( );
44  }

45  DayOfYear::DayOfYear( ) : month(1), day(1)
46  {/*Body intentionally empty.*/}

47  //uses iostream and cstdlib:
48  void DayOfYear::testDate( )
49  {
50      if ((month < 1) || (month > 12))
51      {
52          cout << "Illegal month value!\n";
53          exit(1);
54      }
55      if ((day < 1) || (day > 31))
56      {
57          cout << "Illegal day value!\n";
58          exit(1);
59      }
60  }
```

<Definitions of the other member functions are the same as in Display 6.4.>

SAMPLE DIALOGUE

```
Initialized dates:
February 21
May 1
January 1
date1 reset to the following:
October 31
```

Destructor Functions

- Destructor function is the complement of a constructor function
  - need to clean up the object
  - unlike constructors, only one destructor
  - The compiler automatically generates one if no destructor is declared

- A destructor has the following properties
  - its name = tilde (~) + class name
  - should be public
  - cannot have a return type and any parameter
  - automatically called when the object goes out of scope

Example of Constructor/Destructor

```
class CStr
{
public:
    CStr() { line = NULL; } //A
    CStr(char* cline) { line = cline; } //B
    ~CStr() {}
private:
    char * line;
};
```

```
int main() {
    char* p = new char("Savitch");
    CStr one(p); //call B
    delete [] p;

    return 0;
}
```

Example of Constructor/Destructor (cont.)

- Modified constructor and destructor

```
class CStr
{
public:
    CStr() { line = NULL; } //A
    CStr(char* cline) {
        line = new char [strlen(cline)+1];
        strcpy(line, cline);
    }
    ~CStr() {  //B
        if (line) delete [] line;
        line = NULL;
    }
private:
    char * line;
};
```

Summary

- Review OO concept and programming
  - Procedural programming's problem
  - Three concepts of OOP paradigm
  - What is the relationship between class and objects?
- Class declaration
  - Data members and function members
  - Three access modifiers
  - Two way to define classes and declare objects

Summary (cont.)

- Using objects
  - Three ways to access members in objects
- Member functions
  - Separate declaration form implementation
  - Get and set functions
  - Constructors
  - Destructors

References

- Paul Deitel and Harvey Deitel, “C How to Program” Eight Edition
  - Chapter 16
  - Chapter 17
- Paul Deitel and Harvey Deitel, “C++ How to Program (late objects version)” Seventh Edition
  - Chapter 9: Class
- W. Savitch, “Absolute C++,” Fourth Edition
  - Chapter 6, 7

Another Example: GradeBook (16.3)

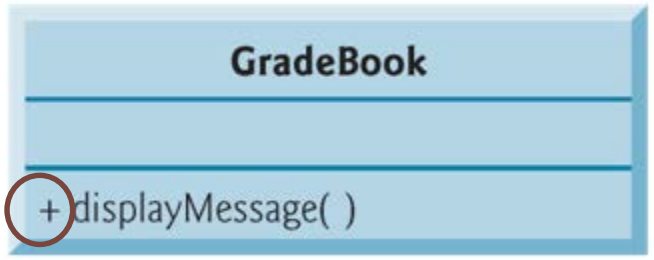
```
// Fig. 16.1: fig16_01.cpp
// Defining a Class with a Member Function.
#include <iostream>
using namespace std;

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

Another Example: GradeBook (16.3) (cont.)

```
// function main begins program execution
int main ()
{
    GradeBook myGradeBook; // create a GradeBook object
    // call object's displayMessage function
    myGradeBook.displayMessage();
}
```

- UML class diagram



Class Name

Data member area

Member Function area

+: public
-: private

Another Example: GradeBook (16.4)

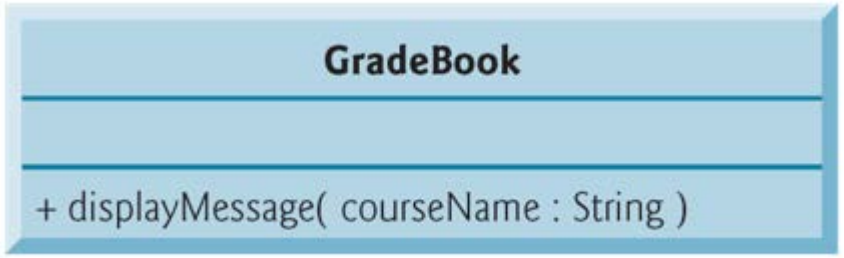
```
// Fig. 16.3: fig16_03.cpp
// Defining a Member Function with a Parameter.
#include <iostream>
#include <string>
using namespace std;

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

Another Example: GradeBook (16.4) (cont.)

```
// function main begins program execution
int main ()
{
    string nameOfCourse;
    GradeBook myGradeBook; // create a GradeBook object
    cout << "Please enter the course name:" << endl;
    getline (cin, nameOfCourse);
    cout << endl;
    // call object's displayMessage function
    myGradeBook.displayMessage(nameOfCourse);
}
```

- UML class diagram



Another Example: GradeBook (16.5)

```
// Fig. 16.5 : fig16_05.cpp
// Defining set and get Member Functions.
#include <iostream>
#include <string>
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
    // function that sets the course name
    void setCourseName(string name)
    {
        courseName = name;
    }
```

Another Example: GradeBook (16.5) (cont.)

```
    // function that gets the course name
    string getCourseName()
    {
        return courseName;
    }
    // function that displays a welcome message
    void displayMessage()
    {
        cout << "Welcome to the grade book for\n"
             << getCourseName() << "!" << endl;
    }
private:
    string courseName;
};
```

Another Example: GradeBook (16.5) (cont.)

```
// function main begins program execution
int main ()
{
    string nameOfCourse;
    GradeBook myGradeBook; // create a GradeBook object
    cout << "Initial course name is "
         << myGradeBook.getCourseName() << endl;
    cout << "Please enter the course name:" << endl;
    getline (cin, nameOfCourse);
    // set the course name
    myGradeBook.setCourseName(nameOfCourse);
    cout << endl;
    // call object's displayMessage function
    myGradeBook.displayMessage();
}
```

Another Example: GradeBook (16.5) (cont.)

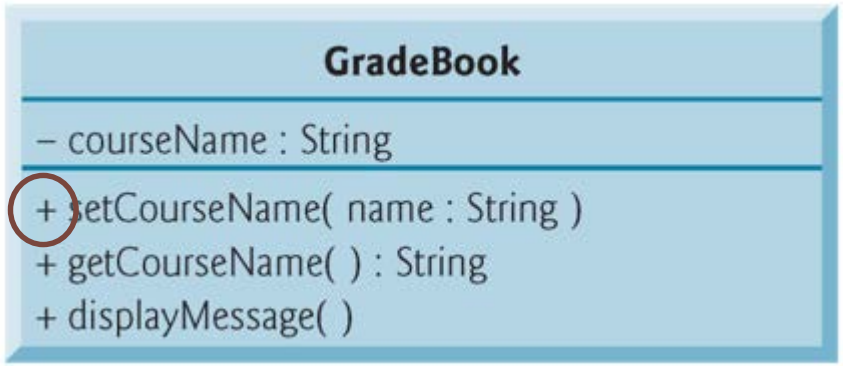
- Screen output

```
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!
```

- UML class diagram



Class Name

Data member area

Member Function area

+: public
-: private

Another Example: GradeBook (16.6)

```
// Fig. 16.7 : fig16_07.cpp
// using the Grade constructor to specify the course name.
#include <iostream>
#include <string>
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
    // constructor initialization course Function that
    // sets the course name
    GradeBook(string name)
    {
        setCourseName(name); // call set function to
        // initialize course to initial courseName
    }
```

Another Example: GradeBook (16.7) (cont.)

```
    // function to set the course name
    void setCourseName(string name)
    {
        courseName = name;
    }
    // function to get the course name
    string getCourseName()
    {
        return courseName;
    }
    // function that displays a welcome message
    void displayMessage()
    {
        cout << "Welcome to the grade book for\n"
             << getCourseName() << "!" << endl;
    }
```

Another Example: GradeBook (16.6) (cont.)

```
private:
    string courseName;
};
Functioning begins program execution
int main ()
{
    // Create two GradeBook objects
    GradeBook myGradeBook1("CS101 Introduction to C++
programming");
    GradeBook myGradeBook2("CS102 Data Structures in C++
");
    // display initial value of courseName
    cout << "gradeBook1 created for course: "
         << gradeBook1.getCourseName()
         << "\ngradeBook2 created for course: "
         << gradeBook2.getCourseName() << endl;
}
```

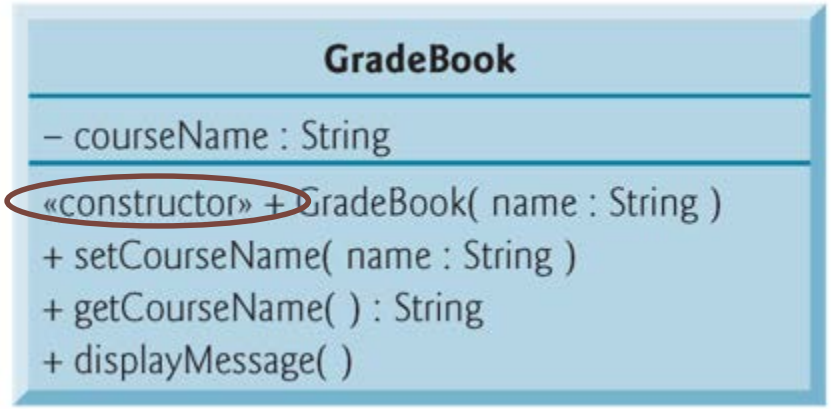
Another Example: GradeBook (16.7) (cont.)

- Screen output

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

- UML class diagram

constructor



Another Example: GradeBook (16.7)

```
// Fig. 16.9 : GradeBook.h
// GradeBook class definition in a separate file from main.
#include <iostream>
#include <string>
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
    // constructor initializes courseName with string
    // supplied as argument
    GradeBook(string name)
    {
        setCourseName(name);
    }
```

Another Example: GradeBook (16.7) (cont.)

```
    // function to set the course name
    void setCourseName(string name)
    {
        courseName = name;
    }
    // function to get the course name
    string getCourseName()
    {
        return courseName;
    }
    // function that displays a welcome message
    void displayMessage()
    {
        cout << "Welcome to the grade book for\n"
             << getCourseName() << "!" << endl;
    }
private:
    string courseName;
};
```

Another Example: GradeBook (16.7) (cont.)

```
// Fig. 16.10 : fig16_10.cpp
// Including class GradeBook from file GradeBook.h
// for use in main file from main.
#include <iostream>
#include "GradeBook.h" // include definition of class
GradeBook
using namespace std;

Functioning begins program execution
int main ()
{
    // Create two GradeBook objects
    GradeBook myGradeBook1("CS101 Introduction to C++
programming");
    GradeBook myGradeBook2("CS102 Data Structures in C++
");
```

Another Example: GradeBook (16.7) (cont.)

```
    // display initial value of courseName
    cout << "gradeBook1 created for course: "
         << gradeBook1.getCourseName()
         << "\ngradeBook2 created for course: "
         << gradeBook2.getCourseName() << endl;
}
```

- Screen output

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

Another Example: GradeBook (16.8)

```
// Fig. 16.11: GradeBook.h
// GradeBook class definition.
#include <string>
using namespace std;
class GradeBook
{
public:
    // constructor that initializes courseName
    GradeBook(string name);
    // function tat sets the course name
    void setCourseName(string);
    // function that gets the course name
    string getCourseName();
    // function that display a welcome message
    void displayMessage();
private;
    string courseName;
};
```

Another Example: GradeBook (16.8) (cont.)

```
// Fig. 16.12: GradeBook.cpp
// GradeBook member-function definitions.
#include <iostream>
#include "GradeBook.h"
using namespace std;

// function to set the course name
GradeBook::GradeBook(string name)
{
    seCourseName(name);
}
// function to set the course name
void GradeBook::setCourseName(string name)
{
    courseName = name;
}
```

Another Example: GradeBook (16.8) (cont.)

```
// function to get the course name
String GradeBook::getCourseName()
{
    return courseName;
}
// function that displays a welcome message
void GradeBook::displayMessage()
{
    cout << "Welcome to the grade book for\n"
         << getCourseName() << "!" << endl;
}
```

Another Example: GradeBook (16.8) (cont.)

```
// Fig. 16.13 : fig16_13.cpp
// GradeBook class demonstration after separating its
// interface from its implementation.
#include <iostream>
// include definition of class GradeBook
#include "GradeBook.h"
using namespace std;

Functioning begins program execution
int main ()
{
    // Create two GradeBook objects
    GradeBook myGradeBook1("CS101 Introduction to C++
programming");
    GradeBook myGradeBook2("CS102 Data Structures in C++
");
```

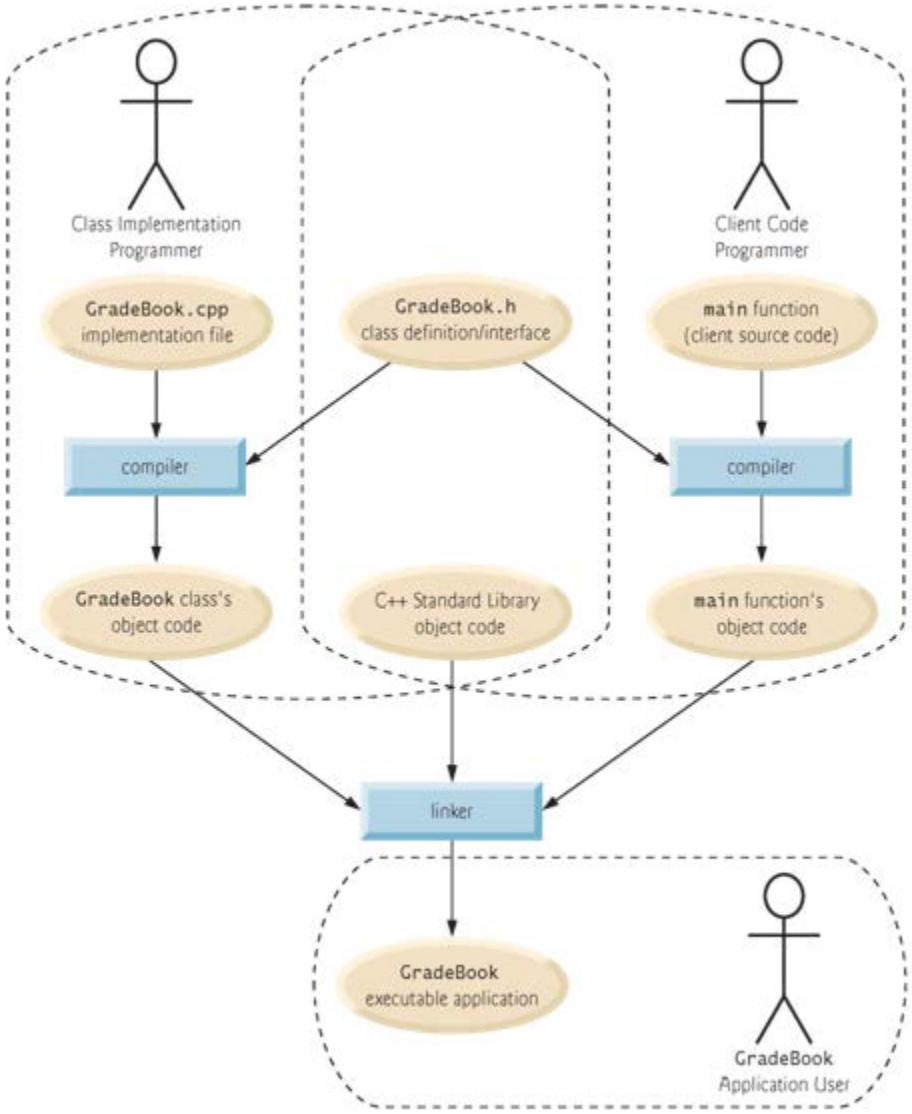
Another Example: GradeBook (16.8) (cont.)

```
    // display initial value of courseName
    cout << "gradeBook1 created for course: "
         << gradeBook1.getCourseName()
         << "\ngradeBook2 created for course: "
         << gradeBook2.getCourseName() << endl;
}
```

- Screen output

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

Compilation and Linking Process



Another Example: GradeBook (16.9)

```
// Fig. 16.15: GradeBook.h
// GradeBook class definition.
#include <string>
using namespace std;
class GradeBook
{
public:
    // constructor that initializes courseName
    GradeBook(string name);
    // function tat sets the course name
    void setCourseName(string);
    // function that gets the course name
    string getCourseName();
    // function that display a welcome message
    void displayMessage();
Private:
    string courseName;
};
```

Another Example: GradeBook (16.9) (cont.)

```
// Fig. 16.16: GradeBook.cpp
// GradeBook member-function definitions.
#include <iostream>
#include "GradeBook.h"
using namespace std;

// function to set the course name
GradeBook::GradeBook(string name)
{
    setCourseName(name);
}
// function to get the course name
String GradeBook::getCourseName()
{
    return courseName;
}
```

Another Example: GradeBook (16.9) (cont.)

```
// function to set the course name
// ensures that the course name has at most 25 characters
void GradeBook::setCourseName(string name)
{
    if (name.length() <= 25)
        courseName = name;
    if (name.length() > 25)
    {
        // start at 0, length of 25
        courseName = name.substr(0, 25);
        cout << "Name \"" << name << "\n" exceeds
                maximum length (25).\n"
             << "Limiting courseName to first 25
                characters.\n" << endl;
    }
}
```

Another Example: GradeBook (16.9) (cont.)

```
// function that displays a welcome message
void GradeBook::displayMessage()
{
    cout << "Welcome to the grade book for\n"
         << getCourseName() << "!" << endl;
}
```

Another Example: GradeBook (16.9) (cont.)

```
// Fig. 16.17 : fig16_17.cpp
// illustrate validation.
#include <iostream>
// include definition of class GradeBook
#include "GradeBook.h"
using namespace std;

Functioning begins program execution
int main ()
{
    // Create two GradeBook objects
    GradeBook myGradeBook1("CS101 Introduction to
Programming in C++");
    GradeBook myGradeBook2("CS102 C++ Data Structures");
```

Another Example: GradeBook (16.9) (cont.)

- Screen output

```
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
```