مبانی برنامه نویسی
به زبان سی

۲۱ و ۲۳ دی ۱۳۹۹

جلسه های ۲۹ و ۳۰

ملکی مجد

مباحث این هفته:

- شی گرایی (مقدماتی)
  - تعریف کلاس
  - UML
  - ساخت شی
  - سازنده کلاس

16.1 Introduction

Employ the basic concepts of object-oriented programming.

- Typically, the C++ programs you develop will consist of function `main` and one or more classes, each containing *data members* and *member functions*.

اگر ترم آینده درس برنامه نویسی پیشرفته به زبان سی شارپ را اخذ کردین، تفاوت سینتکس وجود دارد که منجر به برخی تفاوت ها در نوشتار کد خواهد بود. هر چند مفاهیم کلاس و شی و موارد دیگر یکسان است.

16.2 Classes, Objects, Member Functions and Data Members

- Suppose you want to drive a car and make it go faster by pressing down on its accelerator pedal.
- Before you can drive a car, someone has to *design it and build it*.
- A car typically begins as engineering drawings that include the design for an accelerator pedal that makes the car go faster.
- The pedal "hides" the complex mechanisms that actually make the car go faster, just as the brake pedal "hides" the mechanisms that slow the car, the steering wheel "hides" the mechanisms that turn the car and so on.
- This enables people with little or no knowledge of how cars are engineered to drive a car easily, simply by using the accelerator pedal, the brake pedal, the steering wheel, the transmission shifting mechanism and other such simple and user-friendly "interfaces" to the car's complex internal mechanisms.

16.2 Classes, Objects, Member Functions and Data Members (cont.)

- A function belonging to a class is called a member function.
  - In a class, you provide one or more member functions that are designed to perform the class's tasks.

- Just as you cannot drive an engineering drawing of a car, you cannot "drive" a class.
  - *You must **create an object of a class** before you can get a program to perform the tasks the class describes.*

- Just as many cars can be built from the same engineering drawing, **many objects** can be built from the **same class**.

16.2 Classes, Objects, Member Functions and Data Members (cont.)

- you send messages to an object—each message is known as a **member-function call** and tells a member function of the object to perform its task.
  - This is often called requesting a service from an object.

16.2 Classes, Objects, Member Functions and Data Members (cont.)

- In addition to the capabilities a car provides, it also has many attributes, such as its color, the number of doors, the amount of gas in its tank, its current speed and its total miles driven.
  - Like the car's capabilities, these attributes are represented as part of a car's design in its engineering diagrams.
  - As you drive a car, these attributes are always associated with the car.
  - Every car maintains its own attributes.
- Similarly, an object has attributes that are carried with the object as it's used in a program.
  - These attributes are specified as part of the object's class.
  - Attributes are specified by the class's **data members**.

16.3 Defining a Class with a Member Function

- Begin with an example that consists of
  - class `GradeBook`
    - that an instructor can use to maintain student test scores, and
  - a `main` function that creates a `GradeBook` object.

- Function `main` uses this object and its member function to display a message on the screen welcoming the instructor to the grade-book program.

```
1  // Fig. 16.1: fig16_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 namespace std;
6
7  // GradeBook class definition
8  class GradeBook
9  {
10 public:
11    // function that displays a welcome message to the GradeBook user
12    void displayMessage()
13    {
14       cout << "Welcome to the Grade Book!" << endl;
15    } // end function displayMessage
16 }; // end class GradeBook
17
18 // function main begins program execution
19 int main()
20 {
21    GradeBook myGradeBook; // create a GradeBook object named myGradeBook
22    myGradeBook.displayMessage(); // call object's displayMessage function
23 } // end main
```

```
Welcome to the Grade Book!
```

Fig. 16.1 | Define class `GradeBook` with a member function `displayMessage`, create a `GradeBook` object and call its `displayMessage` function.

16.3 Defining a Class with a Member Function (cont.)

- Function `main` is always called automatically when you execute a program.
  - Most functions do not get called automatically.
  - You must call member function `displayMessage` explicitly to tell it to perform its task.

16.3 Defining a Class with a Member Function (cont.)

- The keyword `public` is an access specifier.
  - Indicates that the function is "available to the public"
  - that is, it can be called by other functions in the program (such as `main`), and by member functions of other classes (if there are any).
    - Access specifiers are always followed by a colon (`:`).

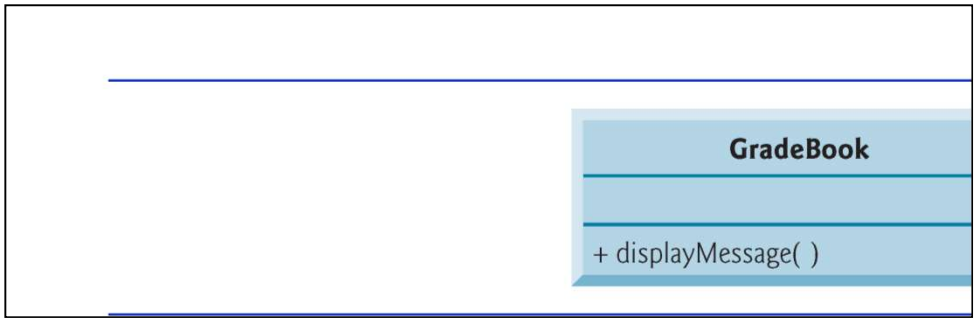
16.3 Defining a Class with a Member Function (cont.)

- Typically, you cannot call a member function of a class until you create an object of that class.
- First, create an object of class `GradeBook` called `myGradeBook`.
  - The variable's type is `GradeBook`.
  - The compiler does not automatically know what type `GradeBook` is—it's a user-defined type.
  - Tell the compiler what `GradeBook` is by including the class definition.
  - Each class you create becomes a new type that can be used to create objects.

- Call the member function `displayMessage-` by using variable `myGradeBook` followed by the dot operator (`.`), the function name `display-Message` and an empty set of parentheses.
  - Causes the `displayMessage` function to perform its task.

16.3 Defining a Class with a Member Function (cont.)

- In the UML, each class is modeled in a UML class diagram as a rectangle with three compartments.



- The top compartment contains the class's name centered horizontally and in boldface type.
- The middle compartment contains the class's attributes,
  - which correspond to data members in C++.
  - Currently empty, because class `GradeBook` does not have any attributes.
- The bottom compartment contains the class's operations,
  - which correspond to member functions in C++.
  - The UML models operations by listing the operation name followed by a set of parentheses.

```
1  // Fig. 16.3: fig16_016.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  #include <string> // program uses C++ standard string class
6  using namespace std;
7
8  // GradeBook class definition
9  class GradeBook
10 {
11 public:
12    // function that displays a welcome message to the GradeBook user
13    void displayMessage( string courseName )
14    {
15       cout << "Welcome to the grade book for\n" << courseName << "!"
16          << endl;
17    } // end function displayMessage
18 }; // end class GradeBook
19
```

Fig. 16.3 | Define class `GradeBook` with a member function that takes a parameter, create a `GradeBook` object and call its `displayMessage` function. (Part 1 of 2.)

```
20  // function main begins program execution
21  int main()
22  {
23     string nameOfCourse; // string of characters to store the course name
24     GradeBook myGradeBook; // create a GradeBook object named myGradeBook
25
26     // prompt for and input course name
27     cout << "Please enter the course name:" << endl;
28     getline( cin, nameOfCourse ); // read a course name with blanks
29     cout << endl; // output a blank line
30
31     // call myGradeBook's displayMessage function
32     // and pass nameOfCourse as an argument
33     myGradeBook.displayMessage( nameOfCourse );
34  } // end main
```

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

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

Fig. 16.3 | Define class `GradeBook` with a member function that takes a parameter, create a `GradeBook` object and call its `displayMessage` function. (Part 2 of 2.)

16.4 Defining a Member Function with a Parameter (cont.)

- A variable of type `string` represents a string of characters.
- A string is actually an object of the C++ Standard Library class `string`.
  - Defined in header file `<string>` and part of namespace `std`.
  - For now, you can think of `string` variables like variables of other types such as `int`.
  - By default, the initial value of a `string` is the so-called empty string, i.e., a string that does not contain any characters.
  - Nothing appears on the screen when an empty string is displayed.

16.4 Defining a Member Function with a Parameter (cont.)

- Library function **`getline`** reads a line of text into a `string`.
- The function call `getline( cin, nameOfCourse )` reads characters (including the space characters that separate the words in the input) from the standard input stream object `cin` (i.e., the keyboard) until the newline character is encountered, places the characters in the `string` variable `nameOfCourse` and discards the newline character.
- When you press *Enter* while typing program input, a newline is inserted in the input stream.

The `<string>` header file must be included in the program to use function `getline`.

16.4 Defining a Member Function with a Parameter (cont.)

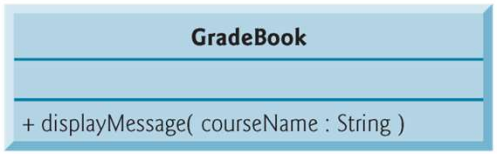


Fig. 16.4 | UML class diagram indicating that class `GradeBook` has a public `displayMessage` operation with a `courseName` parameter of UML type `String`.

- The UML is language independent—it's used with many different programming languages—so its terminology does not exactly match that of C++.

16.5 Data Members, *set Functions and get Functions*

Reminder:

- Variables declared in a function definition's body are known as local variables and can be used only from the line of their declaration in the function to closing right brace (}) of the block in which they're declared.
  - A local variable must be declared before it can be used in a function.
  - A local variable cannot be accessed outside the function in which it's declared.
  - When a function terminates, the values of its local variables are lost.

16.5 Data Members, *set Functions and get Functions (cont.)*

- An object has **attributes** that are carried with it as it's used in a program.
  - Such attributes exist throughout the life of the object.
  - A class normally consists of one or more member functions that manipulate the attributes that belong to a particular object of the class.
- Attributes are represented as variables in a class definition.
  - Such variables are called data members and are declared inside a class definition but outside the bodies of the class's member-function definitions.
- Each object of a class maintains its own copy of its attributes in memory.

16.5 Data Members, *set Functions and get Functions (cont.)*

- A variable that is declared in the class definition but outside the bodies of the class's member-function definitions is a data member.
- Every instance (i.e., object) of a class contains one copy of each of the class's data members.
- A benefit of making a variable a data member is that all the member functions of the class can manipulate any data members that appear in the class definition.

```
1  // Fig. 16.5: fig16_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  #include <string> // program uses C++ standard string class
7  using namespace std;
8
9  // GradeBook class definition
10 class GradeBook
11 {
12 public:
13    // function that sets the course name
14    void setCourseName( string name )
15    {
16       courseName = name; // store the course name in the object
17    } // end function setCourseName
18
19    // function that gets the course name
20    string getCourseName()
21    {
22       return courseName; // return the object's courseName
23    } // end function getCourseName
```

Fig. 16.5 | Defining and testing class `GradeBook` with a data member and *set* and *get* functions. (Part 1 of 3.)

```
24
25      // function that displays a welcome message
26      void displayMessage()
27      {
28         // this statement calls getCourseName to get the
29         // name of the course this GradeBook represents
30         cout << "Welcome to the grade book for\n" << getCourseName() << "!"
31            << endl;
32      } // end function displayMessage
33   private:
34      string courseName; // course name for this GradeBook
35   }; // end class GradeBook
36
37   // function main begins program execution
38   int main()
39   {
40      string nameOfCourse; // string of characters to store the course name
41      GradeBook myGradeBook; // create a GradeBook object named myGradeBook
42
43      // display initial value of courseName
44      cout << "Initial course name is: " << myGradeBook.getCourseName()
45         << endl;
```

Fig. 16.5 | Defining and testing class `GradeBook` with a data member and *set* and *get* functions. (Part 2 of 3.)

```
46
47      // prompt for, input and set course name
48      cout << "\nPlease enter the course name:" << endl;
49      getline( cin, nameOfCourse ); // read a course name with blanks
50      myGradeBook.setCourseName( nameOfCourse ); // set the course name
51
52      cout << endl; // outputs a blank line
53      myGradeBook.displayMessage(); // display message with new course name
54   } // end main
```

```
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!
```

Fig. 16.5 | Defining and testing class `GradeBook` with a data member and *set* and *get* functions. (Part 3 of 3.)

16.5 Data Members, *set Functions and get Functions (cont.)*

- Like `public`, keyword `private` is an access specifier.
- Variables or functions declared after access specifier `private` (and before the next access specifier) are accessible only to member functions of the class for which they're declared.
- The default access for class members is `private` so all members after the class header and before the first access specifier are `private`.
  - The access specifiers `public` and `private` may be repeated, but this is unnecessary and can be confusing.

16.5 Data Members, *set Functions and get Functions (cont.)*

- Declaring data members with access specifier `private` is known as data hiding.
- When a program creates (instantiates) an object, its data members `are` encapsulated (hidden) in the object and can be accessed only by member functions of the object's class.

16.5 Data Members, *set Functions and get Functions (cont.)*

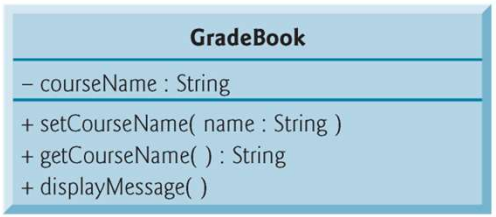


Fig. 16.6 | UML class diagram for class `GradeBook` with a private `courseName` attribute and public operations `setCourseName`, `getCourseName` and `displayMessage`.

- The UML represents data members as attributes by listing the attribute name, followed by a colon and the attribute type.
- The minus sign in the UML is equivalent to the `private` access specifier in C++.

16.6 Initializing Objects with Constructors

- Each class can provide a constructor that can be used to initialize an object of the class when the object is created.
- A constructor is a special member function that must be defined with the same name as the class, so that the compiler can distinguish it from the class's other member functions.
- An important difference between constructors and other functions is that constructors cannot return values, so they cannot specify a return type (not even `void`).
- Normally, constructors are declared `public`.

16.6 Initializing Objects with Constructors (cont.)

- C++ requires a constructor call for each object that is created, which helps ensure that each object is initialized before it's used in a program.
- The constructor call occurs implicitly when the object is created.
- If a class does not explicitly include a constructor, the compiler provides a default constructor—that is, a constructor with no parameters.

```
1  // Fig. 16.7: fig16_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  #include <string> // program uses C++ standard string class
7  using namespace std;
8
9  // GradeBook class definition
10 class GradeBook
11 {
12 public:
13    // constructor initializes courseName with string supplied as argument
14    GradeBook( string name )
15    {
16       setCourseName( name ); // call set function to initialize courseName
17    } // end GradeBook constructor
18
```

Fig. 16.7 | Instantiating multiple objects of the `GradeBook` class and using the `GradeBook` constructor to specify the course name when each `GradeBook` object is created. (Part 1 of 3.)

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

Fig. 16.7 | Instantiating multiple objects of the `GradeBook` class and using the `GradeBook` constructor to specify the course name when each `GradeBook` object is created. (Part 2 of 3.)

```
41
42  // function main begins program execution
43  int main()
44  {
45     // create two GradeBook objects
46     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
47     GradeBook gradeBook2( "CS102 Data Structures in C++" );
48
49     // display initial value of courseName for each GradeBook
50     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
51        << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
52        << endl;
53  } // end main
```

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

Fig. 16.7 | Instantiating multiple objects of the `GradeBook` class and using the `GradeBook` constructor to specify the course name when each `GradeBook` object is created. (Part 3 of 3.)

16.6 Initializing Objects with Constructors (cont.)

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

16.6 Initializing Objects with Constructors (cont.)

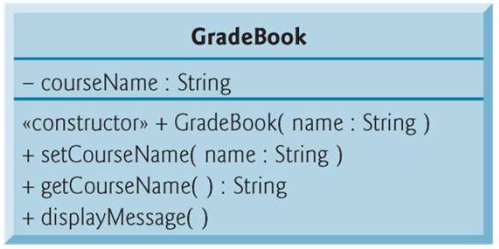


Fig. 16.8 | UML class diagram indicating that class `GradeBook` has a constructor with a `name` parameter of UML type `String`.

- Like operations, the UML models constructors in the third compartment of a class in a class diagram.
- To distinguish a constructor from a class's operations, the UML places the word "constructor" between guillemets (« and ») before the constructor's name.