GLOBAL EDITION

Absolute C++

SIXTH EDITION

Walter Savitch

ALWAYS LEARNING

PEARSON

ABSOLUTE C++

6TH EDITION

GLOBAL EDITION

TIP: (continued)

```
        int getMonthNumber( );
        //Returns 1 for January, 2 for February, etc.
        int getDay( );
    private:
        char firstLetter;//of month
        char secondLetter;//of month
        char thirdLetter;//of month
        int day;
    };
```

In this version, a month is represented by the first three letters in its name, such as `'J'`, `'a'`, and `'n'` for January. The member functions should also be rewritten, of course, but they can be rewritten to behave *exactly* as they did before. For example, the definition of the function `getMonthNumber` might start as follows:

```
int DayOfYear::getMonthNumber( )
{
    if (firstLetter = = 'J' && secondLetter = = 'a'
            && thirdLetter = = 'n')
        return 1;
    if (firstLetter = = 'F' && secondLetter = = 'e'
            && thirdLetter = = 'b')
    return 2;
    . . .
```

This would be rather tedious, but not difficult. ■

Structures versus Classes

Structures are normally used with all member variables public and with no member functions. However, in C++ a structure can have private member variables and both public and private member functions. Aside from some notational differences, a C++ structure can do anything a class can do. Having said this and satisfied the "truth in advertising" requirement, we advocate that you forget this technical detail about structures. If you take this technical detail seriously and use structures in the same way that you use classes, then you will have two names (with different syntax rules) for the same concept. On the other hand, if you use structures as we described them, then you will have a meaningful difference between structures (as you use them) and classes; and your usage will be the same as that of most other programmers.

One difference between a structure and a class is that they differ in how they treat an initial group of members that has neither a public nor a private access specifier. If the first group of members in a definition is not labeled with either `public:` or `private:`, then a structure assumes the group is public, whereas a class would assume the group is private.

Classes and Objects

A class is a type whose variables can have both member variables and member functions. The syntax for a class definition is given as follows.

SYNTAX

```
class Class_Name
{
                .
public:
     Member_Specification_N+1
     Member_Specification_N+2             Public members
                .
                .
                .
private:
     Member_Specification_1
     Member_Specification_2               Private members
                .
                .
                .
     Member_Specification_N
};                Do not forget this semicolon.
```

Each *Member_Specification_i* is either a member variable declaration or a member function declaration (prototype).

Additional `public:` and `private:` sections are permitted. If the first group of members does not have either a `public:` or a `private:` label, then it is the same as if there were a `private:` before the first group.

EXAMPLE

```
class Bicycle
{
public:
     char getColor( );
     int numberOfSpeeds( );
     void set(int theSpeeds, char theColor);
private:
     int speeds;
     char color;
};
```

Once a class is defined, an object variable (variable of the class type) can be declared in the same way as variables of any other type. For example, the following declares two object variables of type `Bicycle`:

```
Bicycle myBike, yourBike;
```

TIP: Thinking Objects

If you have not programmed with classes before, it can take a little while to get the feel of programming with them. When you program with classes, data rather than algorithms take center stage. It is not that there are no algorithms. However, the algorithms are made to fit the data, as opposed to designing the data to fit the algorithms. It is a difference in point of view. In the extreme case, which is considered by many to be the best style, you have no global functions at all, only classes with member functions. In this case, you define objects and how the objects interact, rather than algorithms that operate on data. We will discuss the details of how you accomplish this throughout this book. Of course, you can ignore classes completely or relegate them to a minor role, but then you are really programming in C, not C++. ■

Self-Test Exercises

16. When you define a C++ class, should you make the member variables public or private? Should you make the member functions public or private?

17. When you define a C++ class, what items are considered part of the interface? What items are considered part of the implementation?

Chapter Summary

- A structure can be used to combine data of different types into a single (compound) data value.
- A class can be used to combine data and functions into a single (compound) object.
- A member variable or a member function for a class can be either public or private. If it is public, it can be used outside the class. If it is private, it can be used only in the definition of a member function.
- A function can have formal parameters of a class or structure type. A function can return values of a class or structure type.
- A member function for a class can be overloaded in the same way as ordinary functions are overloaded.
- When defining a C++ class, you should separate the interface and implementation so that any programmer who uses the class need only know the interface and need not even look at the implementation. This is the principle of encapsulation.

Answers to Self-Test Exercises

1. a. `double`

   b. `double`

   c. illegal—cannot use a structure tag instead of a structure variable

   d. `char`

   e. `CDAccountV2`

2.
   ```
   A $9.99
   A $1.11
   ```

3. A semicolon is missing from the end of the definition of `Stuff`.

4. `A x = {1,2};`

5. a. Too few initializers; not a syntax error. After initialization, `month==12`, `day==21`, and `year==0`. Member variables not provided an initializer are initialized to a zero of the appropriate type.

   b. Correct after initialization. `12==month`, `21==day`, and `1995==year`.

   c. Error: too many initializers.

6.
   ```
   struct EmployeeRecord
   {
       double wageRate;
       int vacation;
       char status;
   };
   ```

7.
   ```
   void readShoeRecord(ShoeType& newShoe)
   {
       cout << "Enter shoe style (one letter): ";
       cin >> newShoe.style;
       cout << "Enter shoe price $";
       cin >> newShoe.price;
   }
   ```

8.
   ```
   ShoeType discount(ShoeType oldRecord)
   {
       ShoeType temp;
       temp.style = oldRecord.style;
       temp.price = 0.90*oldRecord.price;
       return temp;
   }
   ```

9.
```
void DayOfYear::input( )
{
    cout << "Enter month as a number: ";
    cin >> month;
    cout << "Enter the day of the month: ";
    cin >> day;
}
```

10.
```
void Temperature::set(double newDegrees, char newScale)
{
    degrees = newDegrees;
    scale = newScale;
}
```

11. Both the dot operator and the scope resolution operator are used with member names to specify of what class or structure the member name is a member. If `class DayOfYear` is as defined in Display 6.3 and `today` is an object of the class `DayOfYear`, then the member `month` may be accessed with the dot operator: `today.month`. When we give the definition of a member function, the scope resolution operator is used to tell the compiler that this function is the one declared in the class.

12.
```
hyundai.price = 4999.99; //ILLEGAL. price is private.
jaguar.setPrice(30000.97); //LEGAL
double aPrice, aProfit; //LEGAL
aPrice = jaguar.getPrice( ); //LEGAL
aProfit = jaguar.getProfit( ); //ILLEGAL. getProfit is
                               //private.
aProfit = hyundai.getProfit( ); //ILLEGAL. getProfit is
                               //private.
hyundai = jaguar; //LEGAL
```

13. After the change, they would all be legal.

14. All members (member variables and member functions) that are marked `private:` can only be accessed by name in the definitions of member functions (both public and private) of the same class. Members marked `public:` have no restrictions on where they can be used.

15. a. Only one. The compiler warns if you have no `public:` members in a class (or `struct`, for that matter).

    b. None, but we normally expect to find at least one `private:` section in a class.

16. The member variables should all be private. The member functions that are part of the interface should be public. You may also have auxiliary (helping) functions that are only used in the definitions of other member functions. These auxiliary functions should be private.

17. All the declarations of private member variables are part of the implementation. (There should be no public member variables.) All the declarations for public member functions of the class (which are listed in the class definitions), as well as the explanatory comments for these declarations, are parts of the interface.

All the declarations for private member functions are parts of the implementation. All member function definitions (whether the function is public or private) are parts of the implementation.

Programming Projects

1. Write a rating program for a software development firm with the following rating policies.

   a. Core competency – This parameter measures an employee's involvement, innovation, and passion toward their assigned tasks, and is graded on 10 points.

   b. Performance evaluation – This parameter measures an employee's performance over the year, and is graded on 30 points.

   c. Ideation – An employee is measured on their patent submissions, patents awarded, technical papers presented, etc. and is graded on 10 points.

   d. The performance evaluation accounts for 50% of an employee's final rating, core competency accounts for 30%, and ideation accounts for 20%.

   The total points obtained by an employee are summed up and normalized. Employees with points of 80 or more are rated a 1, those between 60 to 79 are rated a 2, those between 50 to 59 are rated a 3, and those less than 50 are rated a 4.

   The program will read an employee's points against the three criteria and output the total points obtained on a scale of 100, and the final rating point. Define and use a structure for the employee records.

2. Define a class for a type called `Circle`. An object of type `Circle` is used to display the area, diameter, and circumference of a circle based on the radius of that object. Include a mutator function that sets the radius to a value given as an argument. Include member function to calculate the area, diameter, and circumference of the circle. Also include a member function that returns the radius of the circle. Embed your class definition in a test program.

3. The type `Point` is a fairly simple data type, but under another name (the template class `pair`) this data type is defined and used in the C++ Standard Template Library, although you need not know anything about the Standard Template Library to do this exercise. Write a definition of a class named `Point` that might be used to store and manipulate the location of a point in the plane. You will need to declare and implement the following member functions:

   a. A member function `set` that sets the private data after an object of this class is created.

   b. A member function to move the point by an amount along the vertical and horizontal directions specified by the first and second arguments.

   c. A member function to rotate the point by 90 degrees clockwise around the origin.

   d. Two `const` inspector functions to retrieve the current coordinates of the point.