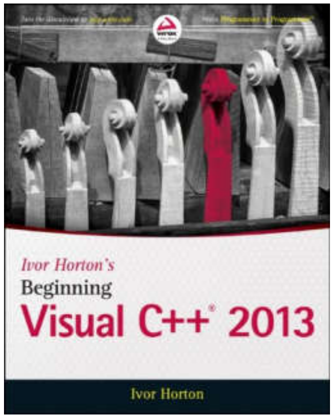


# Chapter 7 (cont.)

# Objects, Classes, and Constructors

# Objects (P.276)

- A `struct` allows you to define a variable representing a composite of several fundamental type variables.
  - You can define a rational number (a,b) which represents a/b, but how do you define a function to perform the addition (a,b)+(c,d)=(b*c+a*d, b*d)?
    - You can define a function add(p,q) to perform the addition.
    - Then consider what you need to do for adding complex numbers a+b*i.*
- An object provides more advanced features:
  - **Encapsulation** – an object contains data and functions that operate on those data
  - Inheritance
  - Polymorphism

# Class

- A **class** is a (user-defined) data type in C++.
  - It can contain data elements of basic types in C++, or of other user-defined types.
  - Just like a `struct`.
  - The keyword `struct` and `class` are almost identical in C++.
  - Let's see an example.

# Example: class CBox (P.279)

```
class CBox
{
    public:
        double m_Length;
        double m_Width;
        double m_Height;
};
```

- When you define `CBox` as a class, you essentially define a new data type.
  - The variables `m_Length`, `m_Width`, `m_Height` which you define are called **data members** of the class.
  - MFC adopts the convention of using the prefix `C` for all class names.
  - MFC also prefixes data members of classes with `m_`.

Microsoft Foundation Classes

# Defining a Class

```
class CBox
{
    public:
        double m_Length;
        double m_Width;
        double m_Height;
};
```

# Accessing Control in a Class

- There are *public* and *private* data members in a class.
  - Public members can be accessed anywhere.
  - Private members can only be accessed by member functions of a class.
  - See Figure 7-6 in next slide.

# Figure 7-6 (P.297)

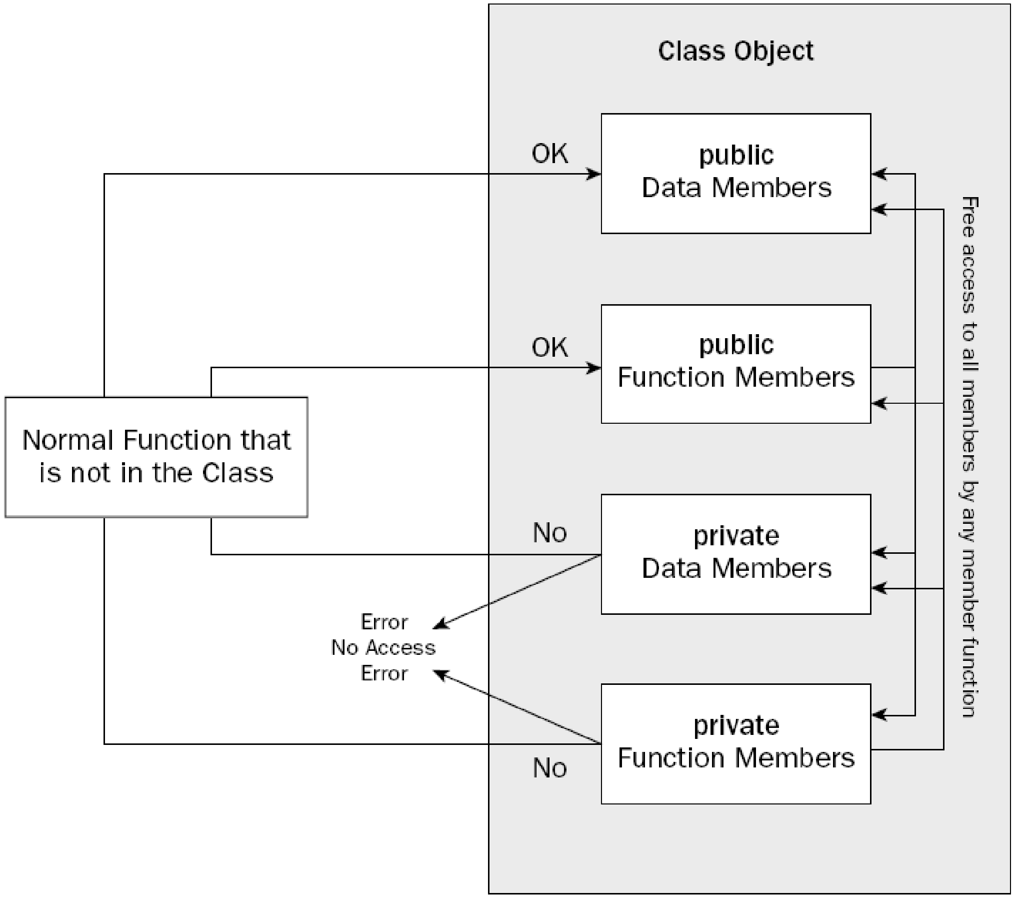


Figure 7-6

# Declaring Objects of a Class (P.280)

```
CBox box1;
CBox box2;
```

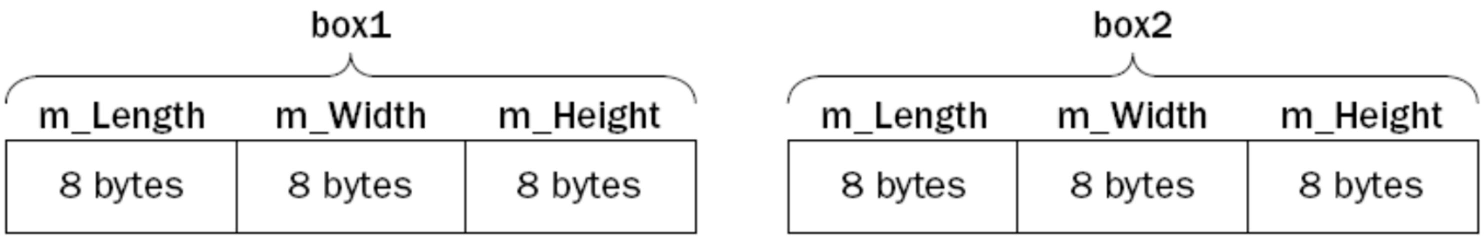


Figure 7-4

# What Does Class Offer More?

- A class can also contain functions.
  - So, a class combines both the definition of the elementary data,
  - and the methods of manipulating these data.
- In this book, we call the data and functions within a class
  - data members
  - member functions

# Member Functions of a Class

- A member function of a class is a function that its definition or its prototype is within the class definition.
  - It operates on any object of the class
  - It has access to all the members of a class, public or private.
- Ex7_03.cpp on P.284
  - `box2.Volume()`
  - There's no need to qualify the names of the class members when you accessing them in member functions.
  - The memory occupied by member functions isn't counted by the `sizeof` operator.

# Positioning a Member Function Definition (1) (Ex7_03a.cpp, P.286)

- For better readability, you may put the definition of a member function outside the class definition, but only put the prototype inside the class.

```
class CBOX
{
    public:
        double m_Length;
        double m_Width;
        double m_Height;
        double Volume(void);
};
```

# Positioning a Member Function Definition (2)

- Now because you put the function definition outside the class, you must tell the compiler that this function belongs to the class CBox.
  - **scope resolution operator ( :: )**

```
// Function to calculate the volume of a box
double CBox::Volume()
{
  return m_Length*m_Width*m_Height;
}
```

# Example: Rational Number

```
#include <iostream>
using std::cout;
using std::endl;

class CRational
{
public:
        int p;
        int q;
};

int main()
{
        CRational a;
        a.p = 1; a.q = 3;

        cout << a.p << '/' << a.q << endl;
        return 0;
}
```

# Rational Number (2)

- Member Functions
  - Print()
  - Reduce()
  - Add()
  - Subtract()
  - Multiply()

# Print()

```
#include <iostream>
using std::cout;
using std::endl;

class CRational
{
public:
        int p;
        int q;

        void Print()
        {
            cout << p << '/' << q << endl;
        }

};

int main()
{
        CRational a;
        a.p = 1; a.q = 3;

        cout << a.p << '/' << a.q << endl;
        a.Print();
        return 0;
}
```

# Multiply()

```
#include <iostream>
using std::cout;
using std::endl;

class CRational
{
public:
    int p;
    int q;

    void Print()
    {
          cout << p << '/' << q << endl;
    }
```

```
          CRational Multiply(CRational b)
          {
              p = p * b.p;
              q *= b.q;
              return *this;
          }

};

int main()
{
          CRational a, b, c;
          a.p = 1; a.q = 3;
          b.p = 2; b.q = 3;

          c = a.Multiply(b);
          c.Print();
          return 0;
}
```

# Initialize Data Members of an Object

- Assign the individual value to each member:
  - Hut1.Left = 70;
  - Hut1.Top = 10;
  - Hut1.Right = 95;
  - Hut1.Bottom = 30;
- It would be great if we have a simpler syntax:
  - RECTANGLE Hut1(70, 10, 95, 30);

# Class Constructors

- A class constructor is a special function which is invoked when a new object of the class is created.
  - You may use the constructor to initialize an object conveniently.
- It always has the same name as the class.
  - The constructor for class CBox is also named CBox().
- It has no return type.
  - You must not even write it as void.

# Ex7_04.cpp on P.288

- Constructor Definition

```
CBox(double lv, double wv, double hv)
{
  cout << "Constructor called for CBox(" << lv << ','
       << wv << ',' << hv << ")." << endl;
  m_Length = lv;    // Set values of data members
  m_Width = wv;
  m_Height = hv;
  }
```

- Object initialization
  - `CBox box1(78.0, 24.0, 18.0);`
  - `CBox cigarBox(8.0, 5.0, 1.0);`

- Observe that the string “Constructor called” was printed out twice in the output.
  - Now that you get the concept of how a constructor works in a class, let us see more variations of constructors.

# The Default Constructor (P.289)

- Try modifying Ex7_04.cpp by adding the following line:
  - `CBox box2;    // no initializing values`
- When you compile this version of the program, you get the error message:
  - `error C2512: 'CBox' no appropriate default constructor available`
- Q: Compare with Ex7_03.cpp (P.284). Why the same line "CBox box2" introduced no troubles at that time?

# The Default Constructor (2)

- In Ex7_03.cpp, you did not declare any constructor, so the compiler generated a default no-argument constructor for you.
- Now, since you supplied a constructor `CBox()`, the compiler assumes that you will take care of everything well.
- You can define a default constructor which actually does nothing:

  -
    ```
    CBox()
    { }
    ```

# Ex7_05.cpp

- The default constructor here only shows a message.
  - This is only an example.  Normally a constructor is used for initialize an object.
- See how the three objects are instantiated.
  - `CBox box1(78.0, 24.0, 18.0);`
  - `CBox box2;`
  - `CBox cigarBox(8.0, 5.0, 1.0);`
- Pay attention to the 6 lines of output messages.

# Assigning Default Parameter Values

- Recall that we may assign default values for function parameters (P.228).
- Put the default values for the parameters in the function header.
  - `int do_it(long arg1=10, long arg2=20);`
- You can also do this for class member functions, including constructors.
- Ex7_06.cpp on P.293

# Hands-on: Function Prototype

- Modify Ex7_06.cpp so that the definition of all member functions (including the Constructor CBox() and Volume()) is placed outside the body of the class definition.
  - Be sure to use the scope resolution operator (::).

# Hint

- The class definition will become:

```
class CBox                    // Class definition at global scope
{
  public:
    double m_Length;              // Length of a box in inches
    double m_Width;              // Width of a box in inches
    double m_Height;             // Height of a box in inches

    // Constructor definition
    CBox(double lv = 1.0, double bv = 1.0, double hv = 1.0);

    // Function to calculate the volume of a box
    double Volume();
};
```

# Using an Initialization List in a Constructor

- It is a common practice to assign *initial values* to data members with constructors.
- Instead of using explicit assignment, you could use a different technique: initialization list:

```
// Constructor definition using an initialisation list
CBox(double lv = 1.0, double bv = 1.0, double hv = 1.0):
                    m_Length(lv), m_Width(bv), m_Height(hv)
{
     cout << endl << "Constructor called.";
}
```

# Private Members of a Class

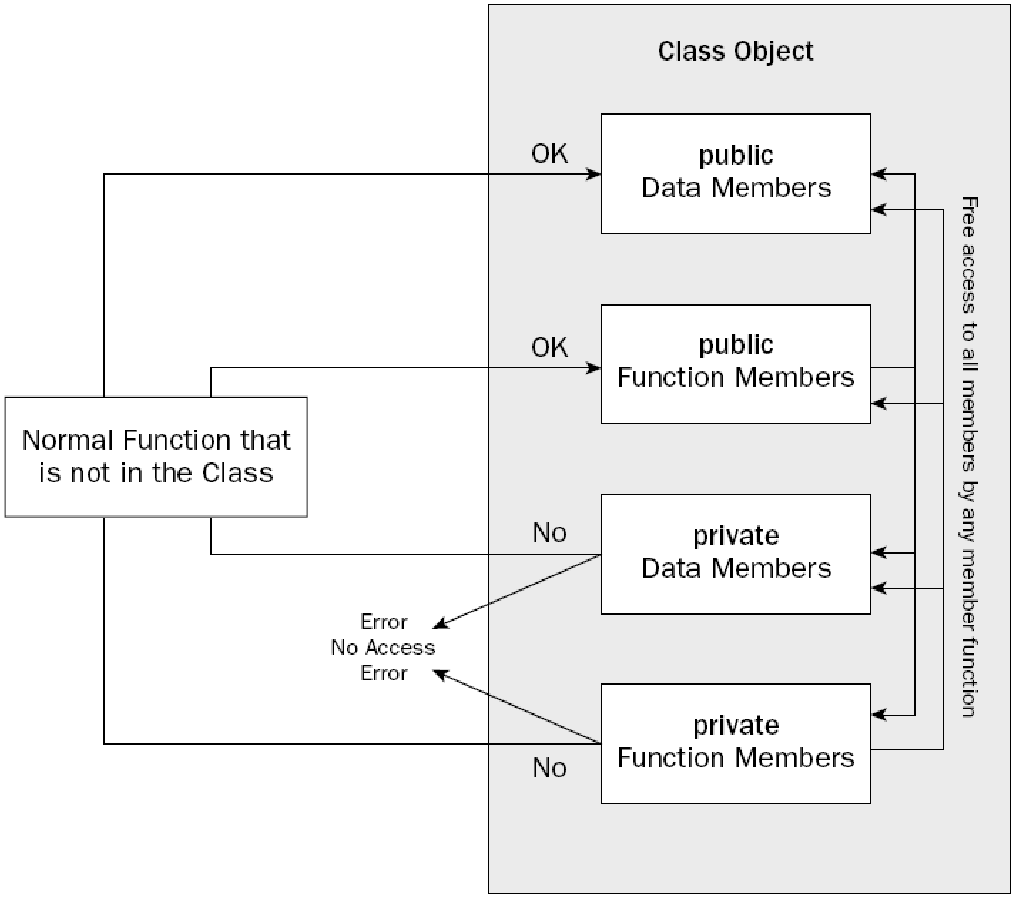


Figure 7-6

# Ex7_07.cpp on P.298

- The definition of the CBox class now has two sections.
  - public section
    - the constructor `CBox()`
    - the member function `Volume()`
  - private section
    - data members `m_Length, m_Width, m_Height`
    - They can only be accessed by member functions of the same class.
    - If you tried to access `m_Length` from `main()`, the compiler would report an error.

# The Copy Constructor

- See the output of Ex7_09.cpp (P.303). The default constructor is only called once.
- How was `box2` created?

- A copy constructor creates an object of a class by initializing it with an existing object of the same class.
- Let us wait until the end of this chapter to see how to implement a copy constructor.

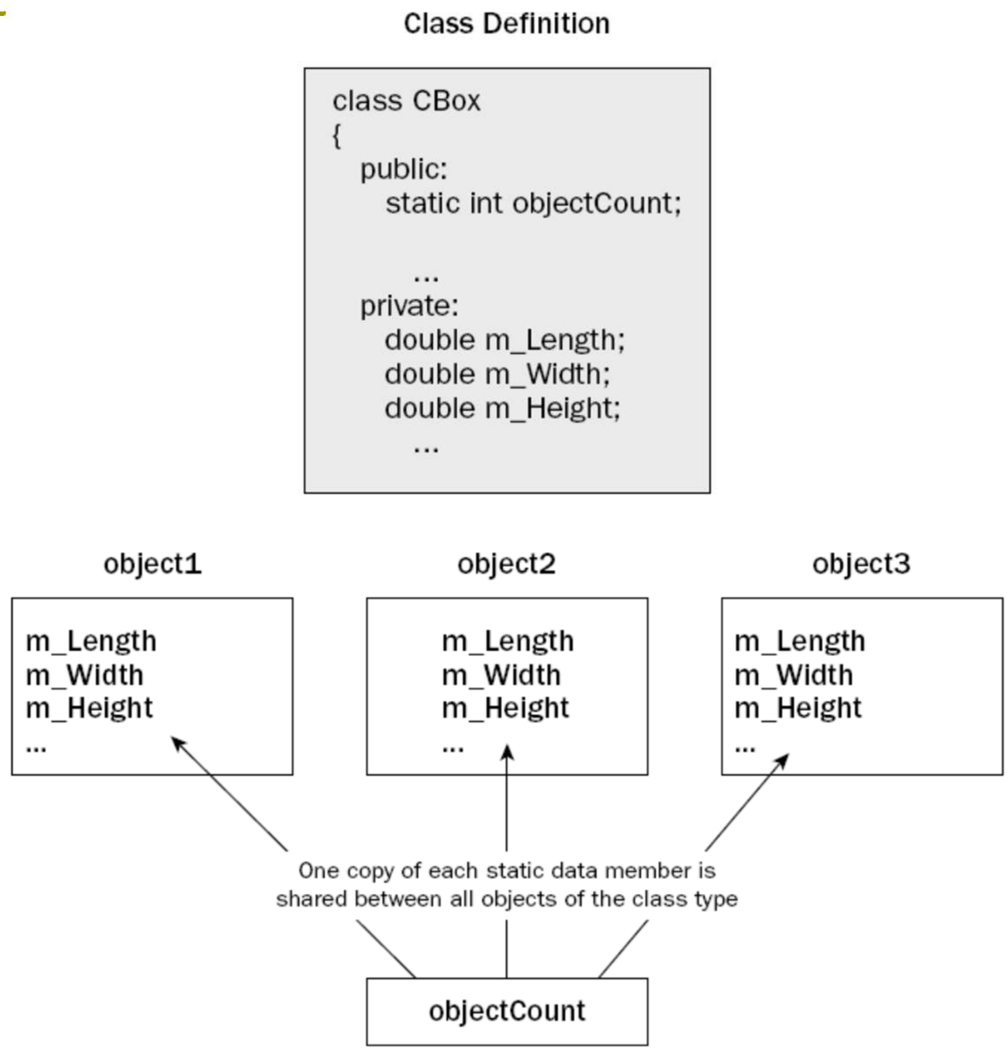
# Arrays of Objects of a Class

- Ex7_11.cpp on P.309

- CBox boxes[5];
- CBox cigar(8.0, 5.0, 1.0);

# Static Data Member of a Class

- When you declare data members of a class to be `static`, the static data members are defined only once and are shared between all objects of the class.
- For example, we can implement a “counter” in this way.



Class Definition

```
class CBox
{
  public:
    static int objectCount;

      ...
  private:
    double m_Length;
    double m_Width;
    double m_Height;
      ...
```

One copy of each static data member is shared between all objects of the class type

Figure 7-7

# How do you initialize the static data member?

- You cannot initialize the static data member in the class definition
  - The class definition is simply a blueprint for objects.  No assignment statements are allowed.
- You don’t want to initialize it in a constructor
  - Otherwise the value will be destroyed whenever a new object is created.

# Counting Instances

- Write an initialization statement of the static data member outside of the class definition:
  - `int CBox::objectCount = 0;`
- Ex7_12.cpp on P.312
  - `static int objectCount;`
  - Declare the count in the public section of CBox class definition.
  - Increment the count in constructors.
  - Initialize the count before main().
    - The static data members exist even though there is no object of the class at all.

# Static Member Functions of a Class

- The static member functions exist, even if no objects of the class exist.

- A static function can be called in relation to a particular object:
  - `aBox.Afunction(10);`
- or with the class name:
  - `CBox::Afunction(10);`

# Pointers to Class Objects

- Declare a pointer to CBox
  - `CBox* pBox = NULL;`
- Store the address of object cigar in pBox
  - `CBox cigar;`
  - `pBox = &cigar;`
- Call the function Volume()
  - `cout << pBox->Volume();`
  - `cout << (*pBox).Volume();`
- In Ex7_10.cpp, the pointer `this` refer to the current object (P.304).

# Implementing a Copy Constructor (P.317)

- Consider writing the prototype of a Copy Constructor like this:
  - `CBox(CBox initB);`
- What happens when this constructor is called?
  - `CBox myBox = cigar;`
- This generates a call of the copy constructor as follows:
  - `CBox::CBox(cigar);`
- This seems to be no problem, until you realize that the argument is passed by value.
  - You end up with an infinite number of calls to the copy constructor.

# Implementing a Copy Constructor (2)

- Use a reference parameter

```
CBox::CBox(const CBox& initB)
{
  m_Length    = initB.m_Length;
  m_Width     = initB.m_Width;
  m_Height    = initB.m_Height;
}
```

- If a parameter to a function is a reference, no copying of the argument occurs when the function is called.
- Declare it as a `const` reference parameter to protect it from being modified from within the function.

# Exercise: Rational Numbers

- Define a class `CRational` with two data members (numerator and denominator), and two member functions.
  - Addition: a/b + c/d = (ad+bc)/bd
  - Reduction (約分) : ac/bc = a/b
  - Store your definition of class CRational in "`rational.h`".
  - Test your class with the following main program:

```
#include "rational.h"

int main()
{
   CRational a(1, 4);
   CRational b(3, 4);
   CRational c = a.Addition(b);
   c.Print();
   c.Reduction(); c.Print();
   CRational d(c); d.Print();
   return 0;
}
```

# Summary

- Encapsulation
  - Data members
  - Member functions
- Public/private members
- Constructors
- Copy Constructors
- Initialization List
- Static data members