

Chapter 7



Defining Your Own Data Types

What Is a `struct`?

- A structure is a user-defined type
 - You define it using the keyword `struct`
 - so it is often referred as a **struct**.
- Compared to the data types we have seen, some real world objects must be described by several items:
 - Time – `hh:mm:ss`
 - Point – `(x,y)`
 - Circle – `(x, y, r)`

Defining a struct

```
struct POINT
{
    float x;
    float y;
};
```

□ Note:

- This doesn't define any variables.
 - It only creates a new type.
- Each line defining an element in the struct is terminated by a semicolon
- A semicolon also appears after the closing brace.

Creating Variables of Type POINT

```
POINT p1, p2;
```

- ▣ If you also want to initializing a struct:

```
POINT p1 =  
{  
    1.0,  
    2.0  
};
```

Accessing the Members of a struct

- Member selection operator (.)
 - `p1.x = 3.0;`
 - `p2.y += 2.0;`

Figure 7-1 on P.326

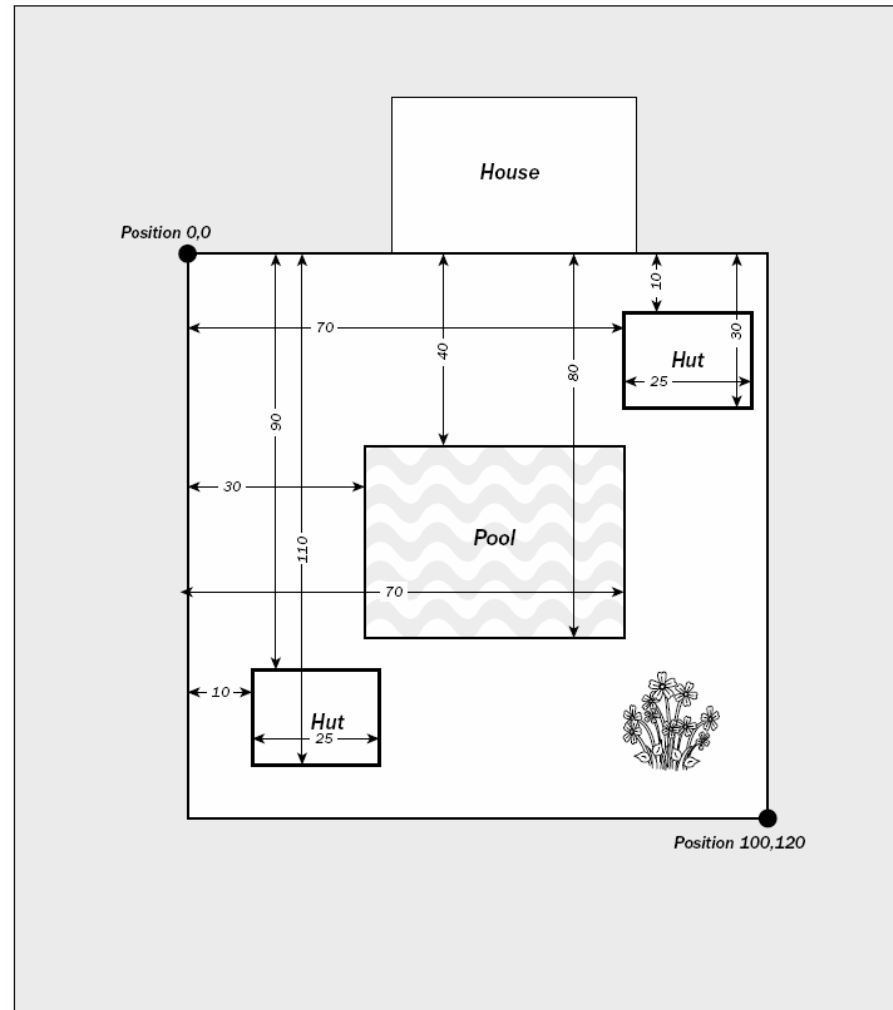


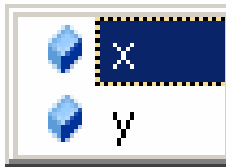
Figure 7-1

Ex7_01.cpp

- `Hut2 = Hut1;`
 - `Hut2.Left = Hut1.Left;`
 - `Hut2.Top = Hut1.Top;`
 - `Hut2.Right = Hut1.Right;`
 - `Hut2.Bottom = Hut1.Bottom;`
- Putting the definition of the struct at global scope allows you to declare a variable of type `RECTANGLE` anywhere in the `.cpp` file.
- Pass by reference

Intellisense Assistance with Structures

```
1 #include <iostream>
2 struct POINT
3 {
4     float x; // X coordinate of the point
5     float y; // Y coordinate of the point
6 };
7
8 int main()
9 {
10
11     POINT p1 = { 1.0, 2.0 };
12     p1.x = 3.0;
13     p1.y += 2.0;
14     p1.
15
16     std::endl;
17 }
18
```



float POINT::x
X coordinate of the point
File: test.cpp

The struct RECT

- ▣ There is a pre-defined structure `RECT` in the header file `windows.h`, because rectangles are heavily used in Windows programs.

```
struct RECT
{
    int left;           // Top left point
    int top;           // coordinate pair

    int right;         // Bottom right point
    int bottom;       // coordinate pair
};
```

Using Pointers with a struct

- `RECT* pRect = NULL;`
 - Define a pointer to RECT

- `pRect = &aRect;`
 - Set pointer to the address of aRect

A struct can contain a pointer

```
struct ListElement
{
    RECT aRect;           // RECT member of structure
    ListElement* pNext;  // Pointer to a list element
};
```

Linked List

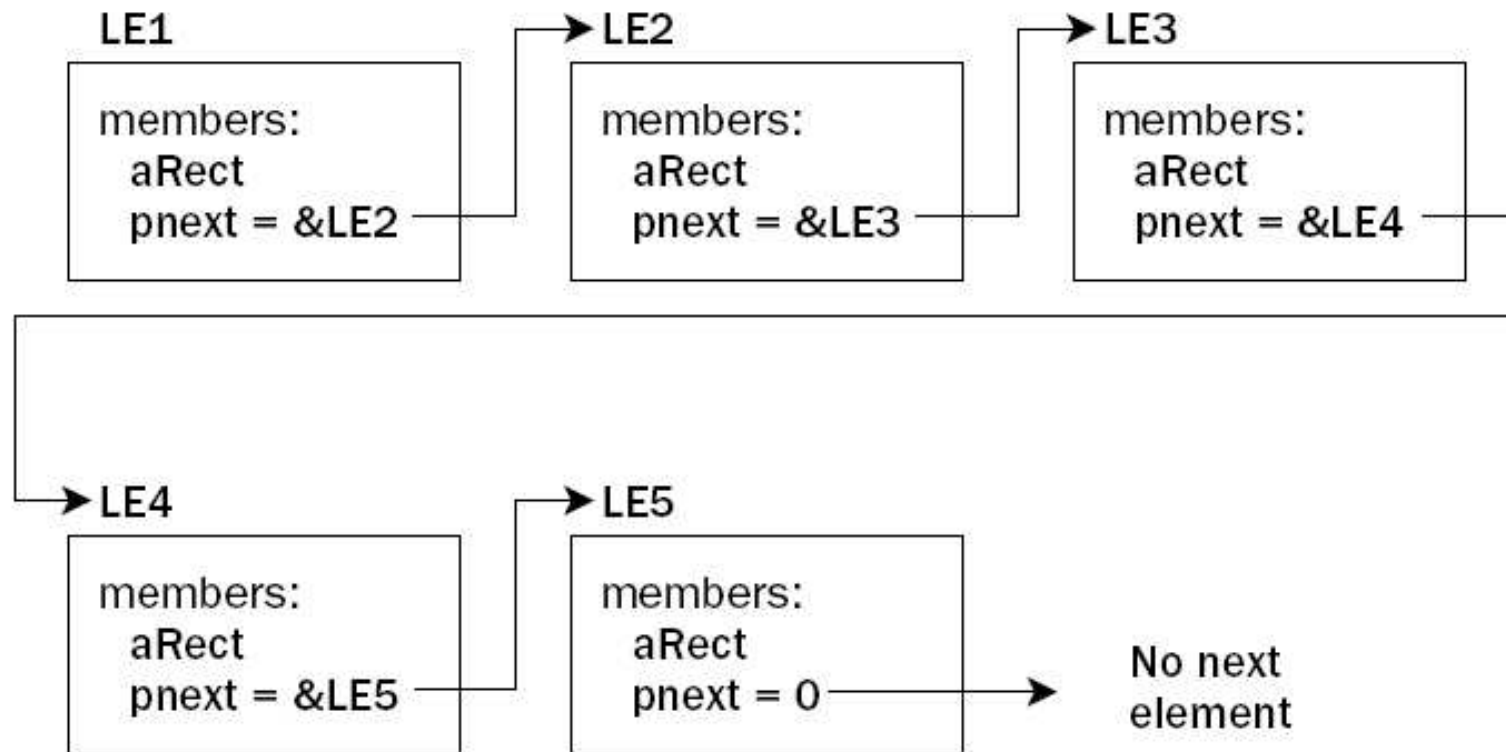


Figure 7-3

Accessing Structure Members through a Pointer

- `RECT aRect = { 0, 0, 100, 100};`
- `RECT* pRect = &aRect;`

- `(*pRect).Top += 10;`
 - The parenthesis to de-reference the pointer are necessary (P.77)

- `pRect->Top += 10;`
 - Indirect member selection operator

HW2

- ❑ Define a struct `Sample` that contains two integer data items.
- ❑ Write a program which declares two object of type `Sample`, called `a` and `b`.
- ❑ Set values for the data items that belong to `a`, and then check that you can copy the values into `b` by simple assignment.

Objects

- ❑ A struct allows you to define a variable representing a composite of several fundamental type variables.
- ❑ An object provides more advanced features:
 - Encapsulation
 - Polymorphism
 - Inheritance

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

```
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_.

What Does Class Offer More?

- A class also can 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

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 on P.351.

Figure 7-6

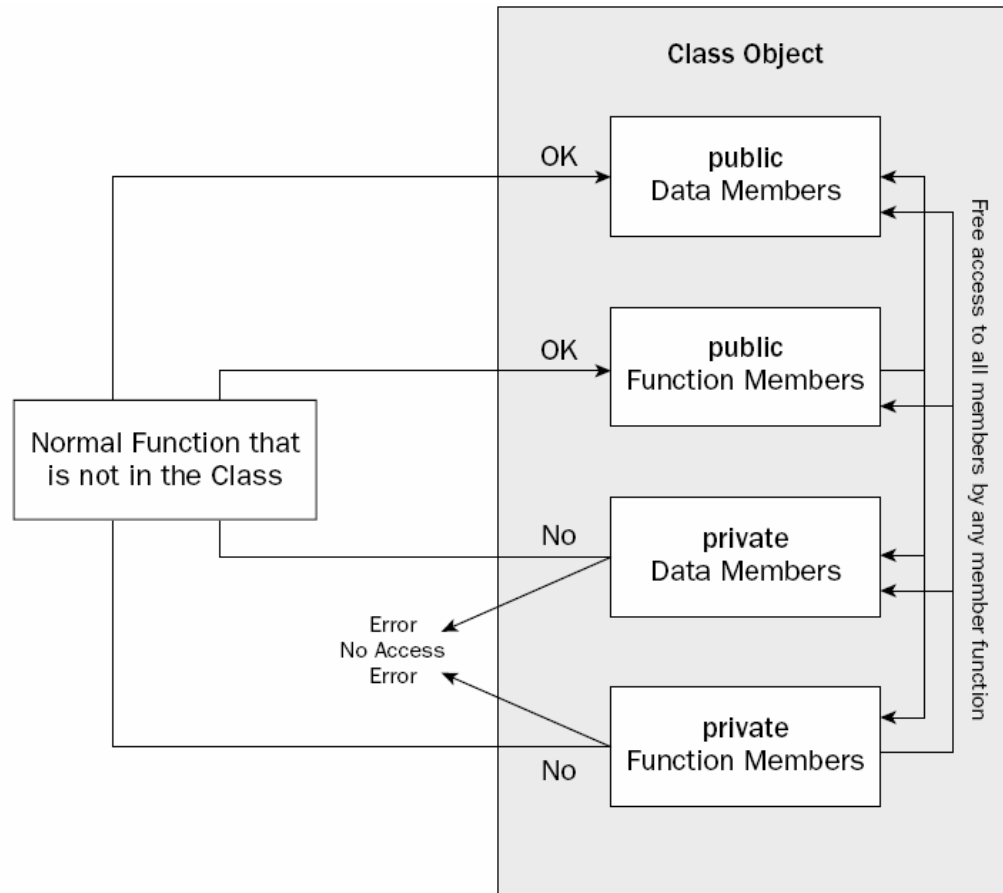


Figure 7-6

Declaring Objects of a Class

```
CBox box1;
```

```
CBox box2;
```

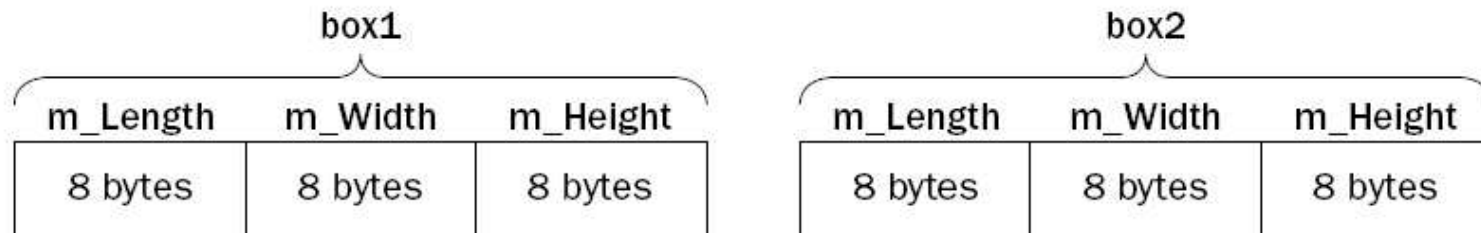


Figure 7-4

Accessing the Data Members of a Class

- `box2.m_Height = 18.0;`
 - direct member selection operator
- `Ex7_02.cpp`
 - The definition of the class appears outside of the function `main()`, so it has **global scope**.
 - You can see the class showing up in the **Class View** tab.

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.339
 - `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)

- ▣ 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;
}
```

HW3

- Modify `Ex7_01.cpp`, so that the yard, the pool, and two huts belong to the type `CIRCLE` instead of `RECTANGLE`.

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.343

❑ Constructor Definition

```
CBox(double lv, double bv, double hv)
{
    cout << endl << "Constructor caled.";
    m_Length = lv;
    m_Width = bv;
    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.

The Default Constructor

- 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_02.cpp. Why the same line "CBox box2" introduced no troubles at that time?

The Default Constructor (2)

- ❑ In `Ex7_02.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 only shows a message.
- ❑ 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.

HW5

- Modify Ex7_06.cpp so that the definition of the Default Constructor is placed outside the body of the class definition.

Assigning Default Parameter Values

- Recall that we may assign default values for function parameters (P.278).
- 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.348

Using an Initialization List in a Constructor

- ▣ 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

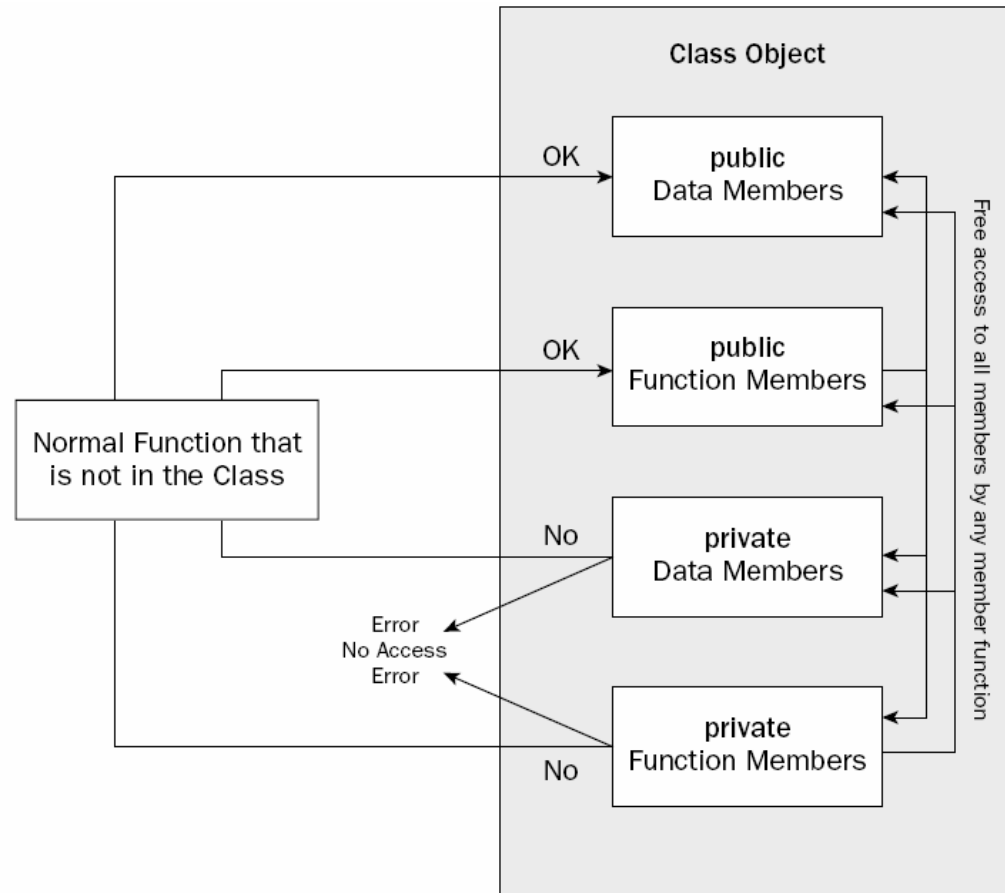


Figure 7-6

Ex7_07.cpp on P.351

- 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`

The Copy Constructor

- ❑ See the output of Ex7_09.cpp. 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.363
- 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.

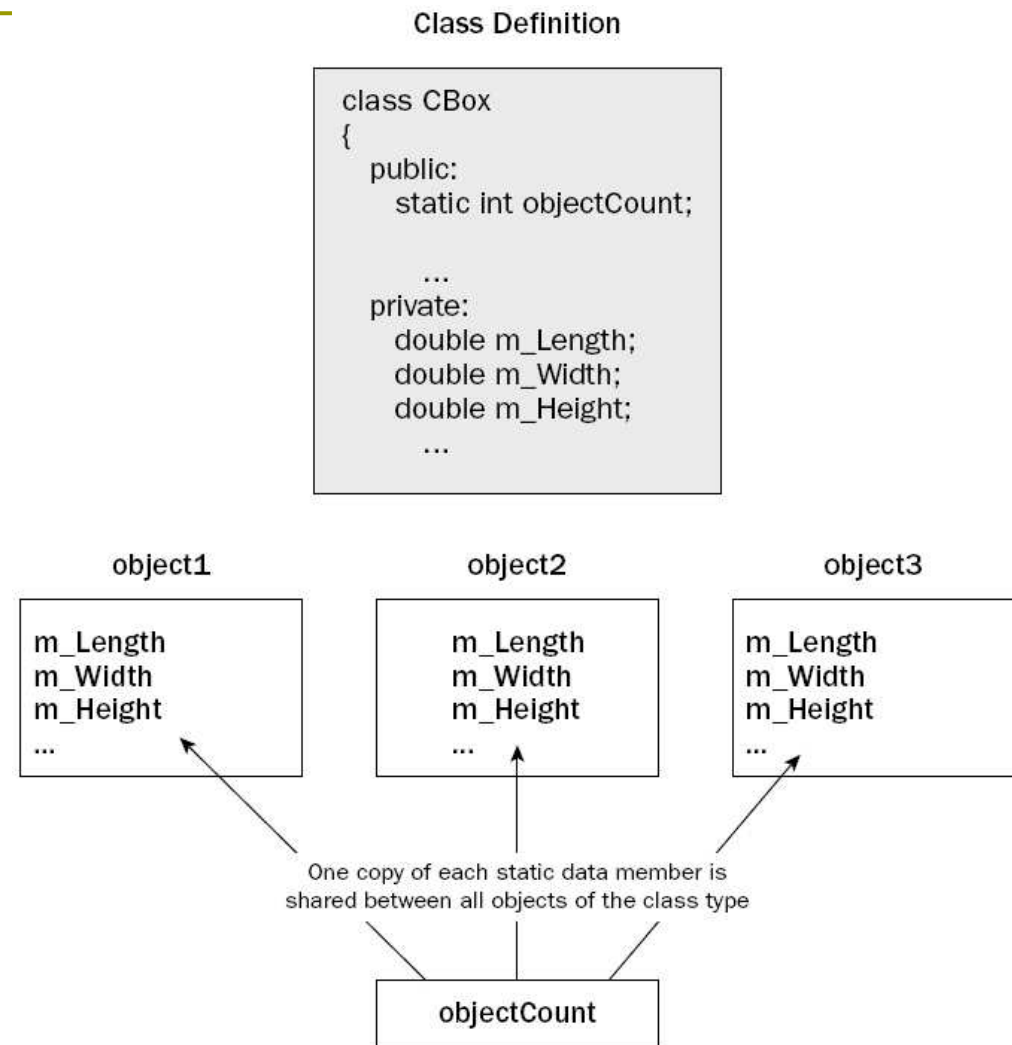


Figure 7-7

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.366
 - `static int objectCount;`
 - 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 = 0;`
- ❑ Store address of object cigar in pBox
 - `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.358).

References to Class Objects

- Remember, a reference acts as an alias for the object.

- Define reference to object cigar
 - `CBox& rcigar = cigar;`

- Output volume of cigar
 - `cout << rcigar.Volume();`

Implementing a Copy Constructor

- ❑ 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.