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.334

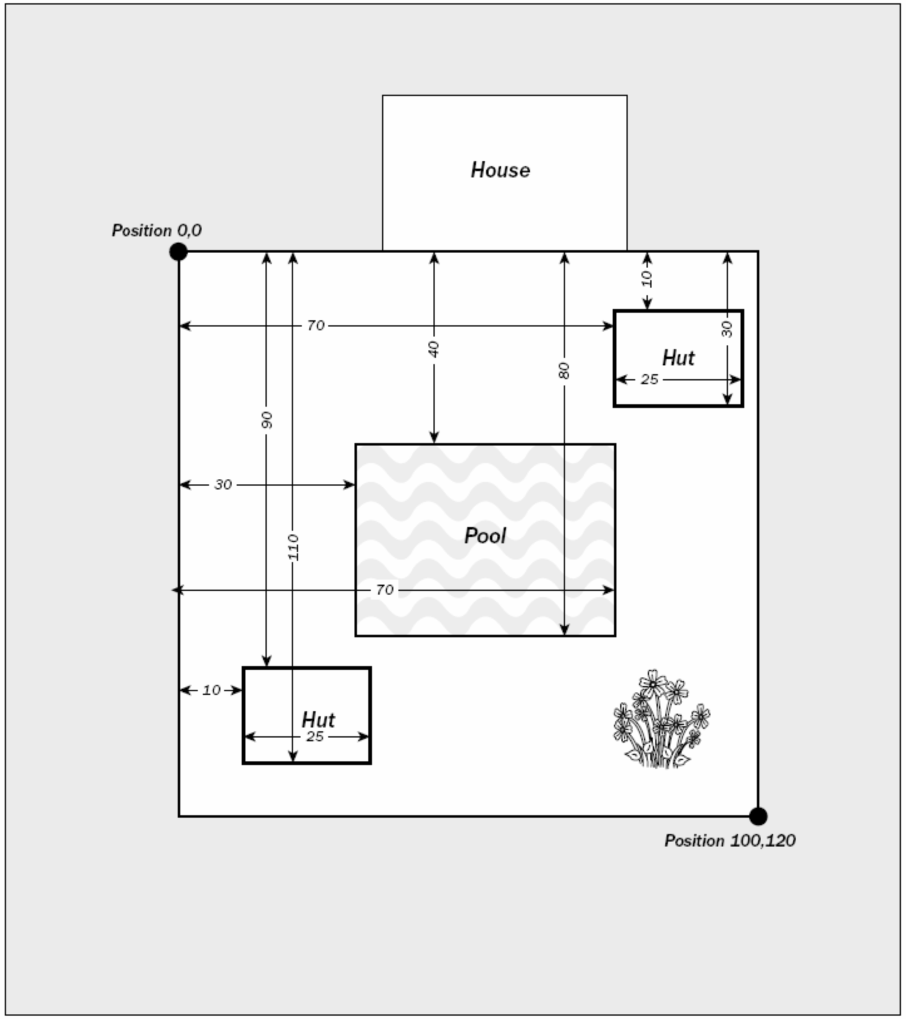

Figure 7-1

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

```cpp
#include <iostream>
        struct POINT
        {
            float x;      // X coordinate of the point
            float y;      // Y coordinate of the point
        };

int main()
{

        POINT p1 = { 1.0, 2.0 };
        p1.x = 3.0;
        p1.y += 2.0;
        p1.

                                  std::endl;
 }
```

x
y

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

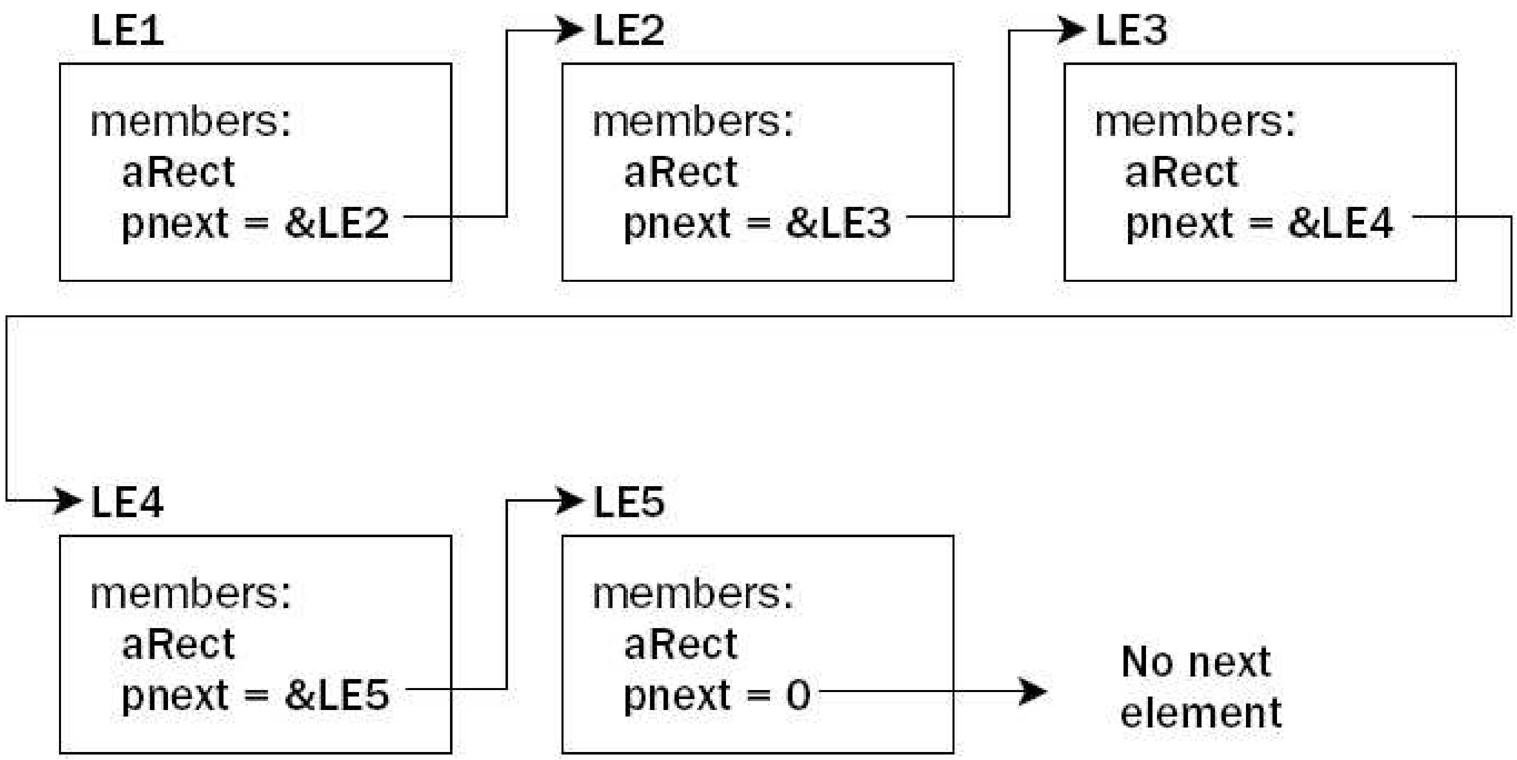

Figure 7-3

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

Exercise

- 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.

Dynamic Memory Allocation (P.194)

- Sometimes depending on the input data, you may allocate different amount of space for storing different types of variables at execution time

```
int n = 0;
cout << "Input the size of the vector - ";
cin >> n;
int vector[n];
```

error C2057: expected constant expression

Why Use Pointers? (P.176)

- Use pointer notation to operate on data stored in an array
- Allocate space for variables dynamically.
- Enable access within a function to arrays, that are defined outside the function

Free Store (Heap)

- To hold a string entered by the user, there is no way you can know in advance how large this string could be.

- Free Store - When your program is executed, there is unused memory in your computer.
- You can dynamically allocate space within the free store for a new variable.

The new Operator

- Request memory for a double variable, and return the address of the space
  - `double* pvalue = NULL;`
  - `pvalue = new double;`
- Initialize a variable created by new
  - `pvalue = new double(9999.0);`
- Use this pointer to reference the variable (indirection operator)
  - `*pvalue = 1234.0;`

The delete Operator

- When you no longer need the (dynamically allocated) variable, you can free up the memory space.
  - `delete pvalue;`
    - Release memory pointed to by pvalue
  - `pvalue = 0;`
    - Reset the pointer to 0

- After you release the space, the memory can be used to store a different variable later.

Allocating Memory Dynamically for Arrays

- Allocate a string of twenty characters
  - `char* pstr;`
  - `pstr = new char[20];`
  - `delete [] pstr;`
    - Note the use of square brackets to indicate that you are deleting an array.
  - `pstr = 0;`
    - Set pointer to null

Dynamic Allocation of Multidimensional Arrays

- Allocate memory for a 3x4 array
  - `double (*pbeans)[4];`
  - `pbeans = new double [3][4];`
- Allocate memory for a 5x10x10 array
  - `double (*pBigArray)[10][10];`
  - `pBigArray = new double [5][10][10];`
- You always use only one pair of square brackets following the delete operator, regardless of the dimensionality of the array.
  - `delete [] pBigArray;`

HW: Linked List

Final Exam

- Date: January 13 (Wednesday)
- Time: 14:10-17:00
- Place: TC-113

- Scope: Chapter 2-7 of Ivor Horton's Beginning Visual C++ 2008
  - CLR programming is excluded.
- Open book
- Turn off computer & mobile phone

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.359.

Figure 7-6

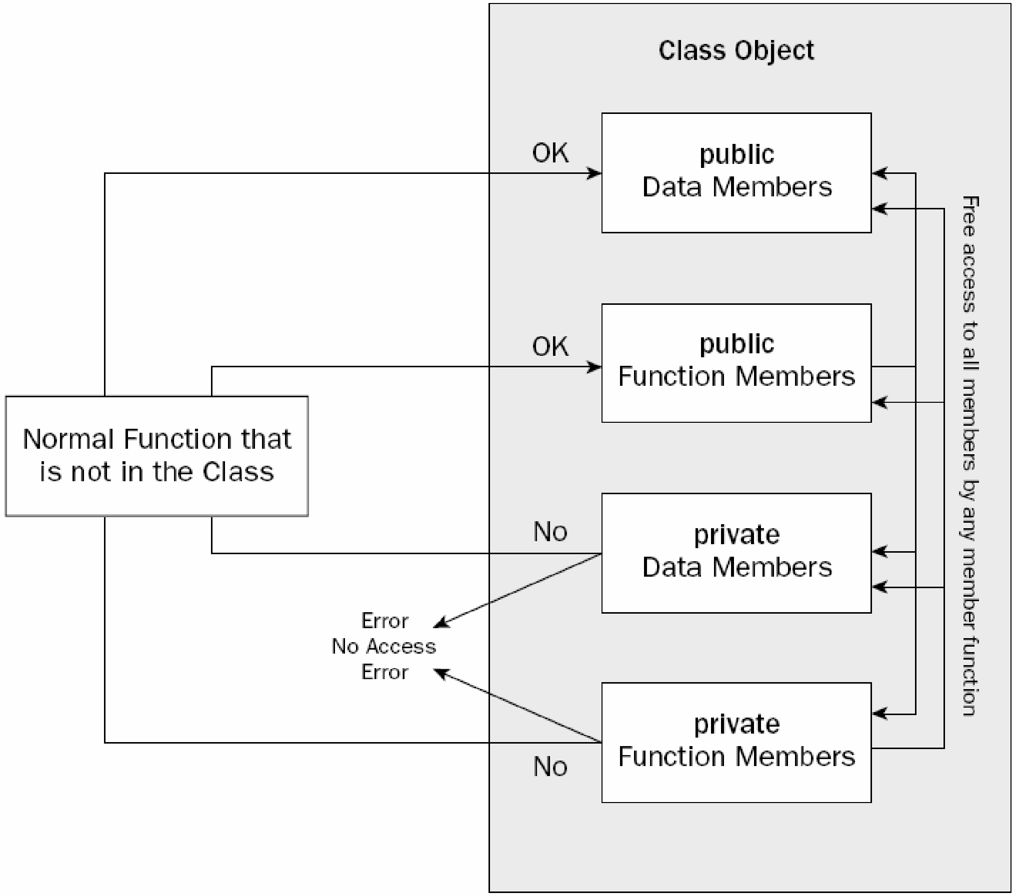

Figure 7-6

Declaring Objects of a Class

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

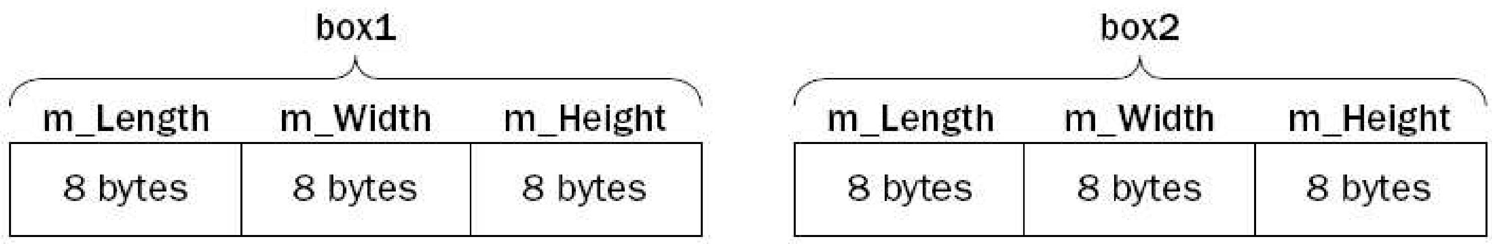

Figure 7-4

P.344

Accessing the Data Members of a Class

- box2.m_Height = 18.0;
  - direct member selection operator
- Ex7_02.cpp (P.345)
  - 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.347
  - 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;
}
```

HW1

- 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.351

- Constructor Definition

```
CBox(double lv, double bv, double hv)
{
  cout << endl << “Constructor called.”;
  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 (P.354)

- 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.

HW2

- 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.285).
- 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.356

Using an Initialization List in a Constructor

- Instead of using explicit assignment, you could use a different technique: initialization list (P.358):

```
// Constructor definition using an initialization 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

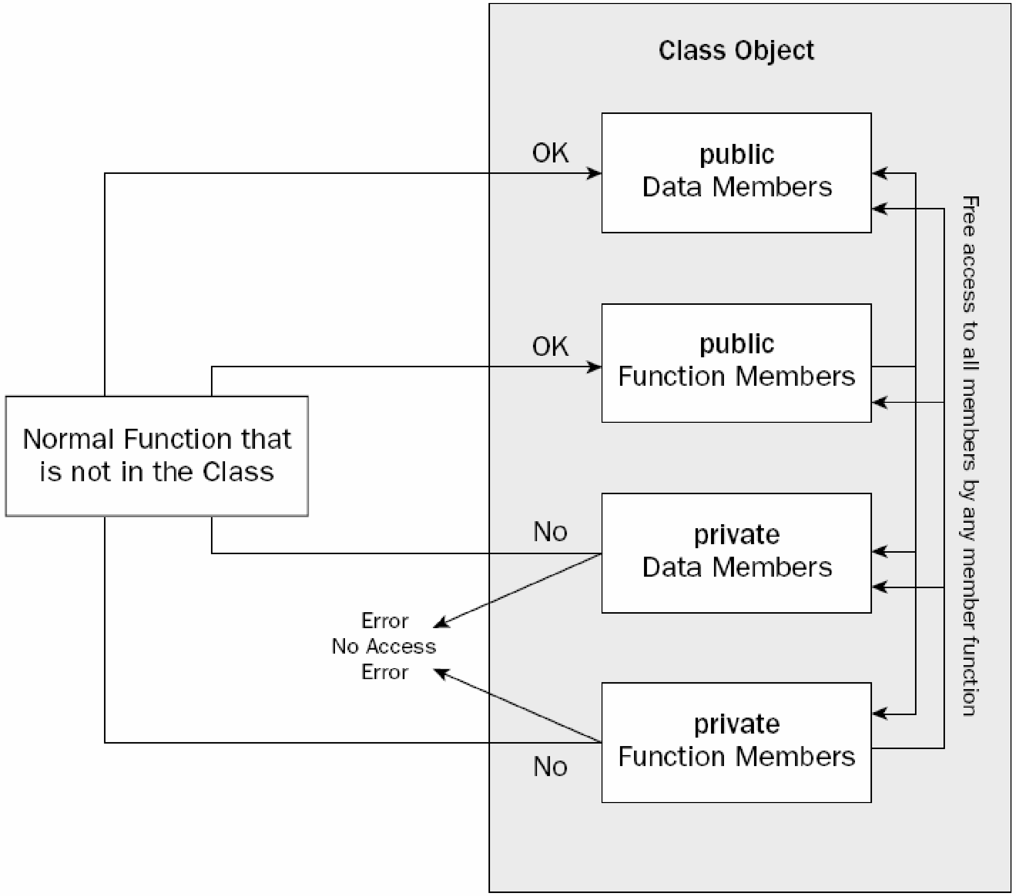

Figure 7-6

Ex7_07.cpp on P.359

- 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 (P.364). 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.371

- 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.

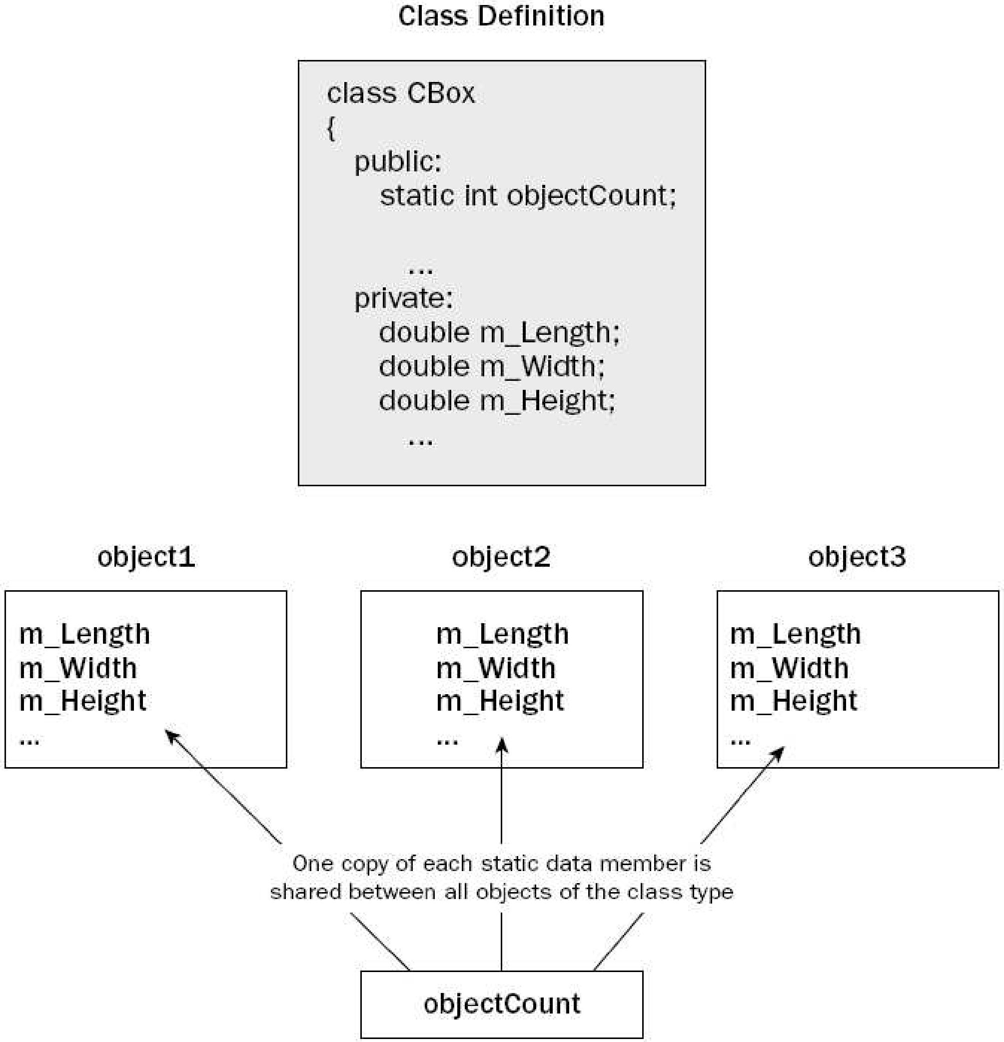

Figure 7-7

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.374
  - `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.366).

References to Class Objects

- Remember, a reference acts as an alias for the object (P.199).

- 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.