1.1 Introduction the class

This chapter introduces the class. The class is the foundation of C++'s support for object-oriented programming, and is at the core of many of its more advancedfeatures. The class is C++'s basic unit of encapsulation and it provides the mechanism by which objects are created.

1.2 ClassFundamentals

Let's begin by defining the terms class and object. A class defines a new data type that specifies the form of an object. A class includes both data and the code that will operate on that data. Thus, a class links data with code. C++ uses a class specification to construct objects. Objects are instances of a class. Therefore, a class is essentially a set of plans that specify how to build an object. It is important to be clear on one issue:

A class is a logical abstraction. It is not until an object of that class has been created that a physical representation of that class exists in memory.

When you define a class, you declare the data that it contains and the code that operates on that data. Although very simple classes might contain only code or only data, most real-world classes contain both. Within a class, data is contained in variables and code is contained in functions. Collectively, the functions and variables that constitute a class are called members of the class. Thus, a variable declared within a class is calleda member variable, and a function declared within a class is called a member function. Sometimes the term instance variable is used in place of member variable.

1.3 The General Form of a class

All classes are declared in a fashion similar to the queue class just described. The general form of a class declaration is shown here.

```
class  class-name {
private data  and functions
public:
public data  and functions
} object-list;
```

A class is created by using the keyword class. Here class-name specifies the name of the class. This name becomes a new type name that can be used to create objects of the class. You can also create objects of the class by specifying them immediately after the class declaration in object-list, but this is optional. Once a class has been declared, objects can be created where needed.Consider the following simple class:

```
// Demonstrate class member access.
#include <iostream.h>
class myclass
{
    int a; // private data
public:
    int b; // public data
    void setab(int i); // public functions
    intgeta();
    void reset();
};
void myclass::setab(int i)
{
  a = i; // refer directly to a
  b = i*i; // refer directly to b
}
intmyclass::geta()
{
    return a; // refer directly to a
}
void myclass::reset()
{
```

```
    // call setab() directly
    setab(0); // the object is already known
}
int main()
{
  myclassob;
  ob.setab(5); // set ob.a and ob.b
  cout<< "ob after setab(5): ";
  cout<<ob.geta() << ' ';
  cout<<ob.b;  //  can  access  b  because  it  is
public
  cout<< '\n';

  ob.b = 20; // can access b because it is
public
  cout<< "ob after ob.b=20: ";
  cout<<ob.geta() << ' ';
  cout<<ob.b;
  cout<< '\n';

  ob.reset();
  cout<< "ob after ob.reset(): ";
  cout<<ob.geta() << ' ';
  cout<<ob.b;
  cout<< '\n';

  return 0;
}
```

This program produces the following output:

```
ob after setab(5): 5 25
ob after ob.b=20: 5 20
ob after ob.reset(): 0 0
```

Let's look carefully at how the members of myclass are accessed. First, notice the way that setab( ) assigns values to the member variables a and b using the lines of code shown here.

```
a = i; // refer directly to a
b = i*i; // refer directly to b
```

Because it is a member function, setab( ) can refer to a and b directly, without explicit reference to an object, and without the use of the dot operator. As explained earlier,a member function is always invoked relative to an object. Once this invocation has occurred, the object is known. Thus, within a member function, there is no need to specify the object a second time. Therefore, references to a and b will apply to the invoking object's copy of these variables.

Next, notice that b is a public member of myclass. This means that b can be accessed by code outside of myclass. This is demonstrated when b is assigned the value 20 inside main( ) using this line of code.

```
ob.b = 20; // can access b because it is public
```

Because this statement is outside of myclass, b must be accessed through an object(in this case, ob) and by use of the dot operator.Now, notice how reset( ) is called from within main( ), as shown here.

```
ob.reset();
```

Because reset( ) is a public member function, it too can be called from code outside of myclass, through an object (in this case, ob).Finally, examine the code inside reset( ). Since reset( ) is a member function, it can directly refer to other members of the class, without use of the dot operator or object. In this case, it calls setab( ). Again, because the object is already known (because it was used to call reset( )), there is no need to specify it again.

The key point to understand is this: When a member of a class is referred to outside of its class, it must be qualified with

an object. However, code inside a member function can refer to other members of the class directly.