1.4 A Closer Look at Class Member Access

How to access class members is the cause of considerable confusion for beginners. For this reason, we will take a closer look at it here.The following class defines a type called queue, which will be used to implement a queue. (A queue is a first-in, first-out list.)

```
// This creates the class queue.
class queue {
    int q[100];
    intsloc, rloc;
public:
    void init();
    void qput(int i);
    intqget();
};
```

Let's look closely at this class declaration.All members of queue are declared within its class statement. The member variables of queue are q, sloc, and rloc. The member functions are init( ), qput( ), and qget( ).

A class can contain private as well as public members. By default, all items defined ina class are private. For example, the variables q, sloc, and rloc are private. This means that they can be accessed only by other members of the queue class, and not by any other part of your program. This is one way encapsulation is achieved—you can tightly control access to certain items of data by keeping them private. Although there are none in this example, you can also define private functions, which can be called only by other members of the class.

To make parts of a class public (i.e., accessible to other parts of your program), you must declare them after the public keyword. All variables or functions defined after the public specifier are accessible by all other functions in your program.

Therefore, in queue, the functions init( ), qput( ), and qget( ) are public. Typically, your program will access the private members of a class through its public functions. Notice that the public keyword is followed by a colon.

Keep in mind that an object forms a bond between code and data. Thus, a member function has access to the private elements of its class. This means that init( ), qput( ), and qget( ) have access to q, sloc, and rloc. To add a member function to a class, specify its prototype within the class declaration. Once you have defined a class, you can create an object of that type by using the class name. A class name becomes a new type specifier. For example, the following statement creates two objects called Q1 and Q2 of type queue:

```
queue Q1, Q2;
```

When an object of a class is created, it will have its own copy of the member variables that comprise the class. This means that Q1 and Q2 will each have their own, separate copies of q, sloc, and rloc. Thus, the data associated with Q1 is distinct and separate from the data associated with Q2.

To access a public member of a class through an object of that class, use the dot operator, just the way you do when operating on a structure. For example, to output Q1's value of sloc, use the following statement.

```
cout<< Q1.sloc;
```

Let's review: In C++, a class creates a new data type that can be used to create objects. Specifically, a class creates a logical framework that defines a relationship between its members. When you declare a variable of a class, you are creating an object. An object has physical existence, and is a specific instance of a class. (That is, an object occupies memory space,

but a type definition does not.) Further, each object of a class has its own copy of the data defined within that class.

Inside the declaration of queue, prototypes for the member functions are specified. Because the member functions are prototyped within the class definition, they need not be prototyped elsewhere.

To implement a function that is a member of a class, you must tell the compiler to which class the function belongs by qualifying the function name with the class name. For example, here is one way to code the qput( ) function:

```
void queue::qput(int i)
{
if(sloc==100)
{
cout<< "Queue is full.\n";
return;
}
sloc++; q[sloc] = i;
}
```

The **::** is called the scope resolution operator. Essentially, it tells the compiler that this version of qput( ) belongs to the queue class. Or, put differently, :: states that this qput( ) is in queue's scope. Several different classes can use the same function names. The compiler knows which function belongs to which class because of the scope resolution operator and the class name.

Member functions can only be invoked relative to a specific object. To call a member function from a part of your program that is outside the class, you must use the object's name and the dot operator. For example, this calls init( ) on object ob1:

```
queue ob1, ob2;
ob1.init();
```

The invocation ob1.init( ) causes init( ) to operate on ob1's copy of the data. Keep in mind that ob1 and ob2 are two separate objects. This means, for example, that initializing ob1 does not cause ob2 to also be initialized. The only relationship ob1 has with ob2 is that it is an object of the same type.

When one member function calls another member function of the same class, it can do so directly, without using an object and the dot operator. In this case, the compiler already knows which object is being operated upon. It is only when a member function is called by code that is outside the classthat the object name and the dot operator must be used. By the same reasoning, a member function can refer directly to a member variable, but code outside the class must refer to the variable through an object and the dot operator.

The program shown here puts together all the pieces and missing details, and illustrates the queue class:

```
#include <iostream.h>
// This creates the class queue.
class queue {
    int q[100];
    intsloc, rloc;
public:
    void init();
    void qput(int i);
    intqget();
};
// Initialize the queue.
void queue::init()
{
     rloc = sloc = 0;
}
// Put an integer into the queue.
void queue::qput(int i)
{
  if(sloc==100)
```

```
  {
     cout<< "Queue is full.\n";
     return;
  }
  sloc++; q[sloc] = i;
}

// Get an integer from the queue.
int queue::qget()
{
  if(rloc == sloc)
  {
     cout<< "Queue underflow.\n";
     return 0;
  }
  rloc++;
  return q[rloc];
}

int main()
{
queue a, b;  // create two queue objects
a.init();
b.init();

a.qput(10);
b.qput(19);

a.qput(20);
b.qput(1);

cout<< "Contents of queue a: ";
cout<<a.qget() << " ";
cout<<a.qget() << "\n";

cout<< "Contents of queue b: ";
cout<<b.qget() << " ";
cout<<b.qget() << "\n";

return 0;
```

```
}
```

This program displays the following output:

```
Contents of queue a: 10 20
Contents of queue b: 19 1
```

Keep in mind that the private members of a class are accessible only by functions that are members of that class. For example, a statement like

```
a.rloc = 0;
```

could not be included in the main( ) function of the program.