1.5 Constructors and Destructors

It is very common for some part of an object to require initialization before it can be used. For example, consider the queue class, shown earlier in this chapter. Before the queue could be used, the variables rloc and sloc had to be set to zero. This was performed using the function init( ). Because the requirement for initialization isso common, C++ allows objects to initialize themselves when they are created. This automatic initialization is performed through the use of a constructor.A constructor is a special function that is a member of a class and that has the same name as the class. For example, here is how the queue class looks when it is converted to use a constructor for initialization:

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

Notice that the constructor queue( ) has no return type. In C++, constructors do not return values and, therefore, have no return type. (Not even void may be specified.). The queue( ) function is coded as follows:

```
// This is the constructor.
queue::queue()
{
   sloc = rloc = 0;
   cout<< "Queue Initialized.\n";
}
```

Here, the message Queue Initialized is output as a way to

illustrate the constructor. In actual practice, most constructors do not print a message.

An object's constructor is called when the object is created. This means that it is called when the object's declaration is executed. For global objects, the constructor is called when the program begins execution, prior to the call to main( ). For local objects, the constructor is called each time the object declaration is encountered.The complement of the constructor is the destructor. In many circumstances, an object will need to perform some action or series of actions when it is destroyed. Local objects are created when their block is entered, and destroyed when the block is left.

Global objects are destroyed when the program terminates. There are many reasons why a destructor may be needed. For example, an object may need to deallocate memory that it had previously allocated. In C++, it is the destructor that handles deactivation. The destructor has the same name as the constructor, but the destructor's name is preceded by a ~. Like constructors, destructors do not have return types.

Here is the queue class that contains a constructor and destructor. (Keep in mind that the queue class does not require a destructor, so the one shown here is just forillustration.)

```
class queue
{
   int q[100];
   intsloc, rloc;
public:
   queue();  // constructor
   ~queue(); // destructor void qput(int i);
   intqget();
};
// This is the constructor.
queue::queue()
{
   sloc = rloc = 0;
   cout<< "Queue initialized.\n";
}
// This is the destructor.
queue::~queue()
{
     cout<< "Queue destroyed.\n";
}
```

Here is a new version of the queue program that demonstrates the constructor and destructor:

```
#include <iostream>
class queue
{
   int q[100];
   intsloc, rloc;
public:
   queue();  // constructor
   ~queue(); // destructor
   void qput(int i);
   intqget();
};
queue::queue()
{
   sloc = rloc = 0;
   cout<< "Queue initialized.\n";
}
queue::~queue()
```

```
{
     cout<< "Queue destroyed.\n";
}
// 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.qput(10);
   b.qput(19);
   a.qput(20);
   b.qput(1);

   cout<<a.qget() << " ";
   cout<<a.qget() << " ";
   cout<<b.qget() << " ";
   cout<<b.qget() << "\n";

   return 0;
}
```

This program displays the following output:

```
Queue initialized.
```

```
Queue initialized.
10 20 19 1
Queue destroyed.
Queue destroyed.
```

1.6 Parameterized Constructors

A constructor can have parameters. This allows you to give member variables program- defined initial values when an object is created. You do this by passing arguments to an object's constructor. The next example will enhance the queue class to accept an argument that will act as the queue's ID number. First, queue is changed to look like this:

```
// This creates the class queue.
class queue
{
   int q[100];
   intsloc, rloc;
   int who; // holds the queue's ID number 4
public:
   queue(int id); // parameterized constructor
   ~queue(); // destructor
   void qput(int i);
   intqget();
};
```

The variable who is used to hold an ID number that will identify the queue. Its actual value will be determined by what is passed to the constructor in id when a variable of type queue is created. The queue( ) constructor looks like this:

```
// This is the constructor.
queue::queue(int id)
{
sloc = rloc = 0;
who = id;
cout<< "Queue " << who << " initialized.\n";
}
```

To pass an argument to the constructor, you must associate the argument with an object when the object is declared. C++ supports two ways to accomplish this. The first method is illustrated here:

```
queue a = queue(101);
```

This declaration creates a queue called a and passes the value 101 to it. However, this form is seldom used (in this context), because the second method is shorter and more to the point. In the second method, the argument must follow the object's name and must be enclosed between parentheses. For example, this statement accomplishes the same thing as the previous declaration:

```
queue a(101);
```

This is the most common way that parameterized objects are declared. Using this method, the general form of passing arguments to constructors is

```
class-type  var(arg-list);
```

Here, arg-list is a comma-separated list of arguments that are passed to the constructor.The following version of the queue program demonstrates a parameterized constructor:

```
// Use a parameterized constructor.
#include <iostream.h>
// This creates the class queue.
class queue
{
   int q[100];
   intsloc, rloc;
   int who; // holds the queue's ID number
public:
   queue(int id); // parameterized constructor
   ~queue(); // destructor
   void qput(int i);
   intqget();
```

```
};
// This is the constructor.
queue::queue(int id)
{
   sloc = rloc = 0;
   who = id;
   cout<<    "Queue    "    <<    who    <<    "
initialized.\n";
}
// This is the destructor.
queue::~queue()
{
     cout<< "Queue " << who << " destroyed.\n";
}
// 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(1), b(2);// create two queue objects
```

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

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

cout<<a.qget() << " ";
cout<<a.qget() << " ";
cout<<b.qget() << " ";
cout<<b.qget() << "\n";

return 0;
}
```

This program produces the following output:

```
Queue 1 initialized.
Queue 2 initialized.
10 20 19 1
Queue 2 destroyed.
Queue 1 destroyed.
```