Function Objects

A *function object*, or *functor*, is any type that implements operator(). This operator is referred to as the *call operator* or sometimes the *application operator*. The Standard Template Library uses function objects primarily as sorting criteria for containers and in algorithms.
Function objects provide two main advantages over a straight function call. The first is that a function object can contain state. The second is that a function object is a type and therefore can be used as a template parameter.

Creating a Function Object

To create a function object, create a type and implement operator(), such as:

```
class Functor
{
public:
    int operator()(int a, int b)
    {
        return a < b;
    }
};

int main()
{
    Functor f;
    int a = 5;
    int b = 7;
    int ans = f(a, b);
}
```

The last line of the **main** function shows how you call the function object. This call looks like a call to a function, but it is actually calling operator() of the Functor type. This similarity between calling a function object and a function is how the term function object came about.

Function Objects and Containers

The Standard Template Library contains several function objects in the <functional> header file. One use of these function objects is as a sorting criterion for containers. For example, the **set** container is declared as follows:

```
template <
    class Key,
    class Traits=less<Key>,
    class Allocator=allocator<Key> >
class set
```

The second template argument is the function object **less**. This function object returns true if the first parameter passed to it is less than the second parameter passed. Since some containers sort their elements, the container needs a way of comparing two elements, and this is accomplished using the function object. You can define your own sorting criteria for containers by creating a function object and specifying it in the template list for the container.

Function Objects and Algorithms

Another use of functional objects is in algorithms. For example, the **remove_if** algorithm is declared as follows:

```
template<class ForwardIterator, class Predicate>
    ForwardIterator remove_if(
        ForwardIterator _First,
        ForwardIterator _Last,
        Predicate _Pred
    );
```

The last argument to **remove_if** is a function object that returns a boolean value (a *predicate*). If the result of the function object is true, then the element is removed from the container being accessed by the iterators *_First* and *_Last*. You can use any of the function objects declared in the <functional> header for the argument *_Pred* or you can create your own.