6.6. User-Defined Generic Functions

The STL is an extensible framework. This means you can write your own functions and algorithms to process elements of collections. Of course, these operations may also be generic. However, to declare a valid iterator in these operations, you must use the type of the container, which is different for each container type. To facilitate the writing of generic functions, each container type provides some internal type definitions. Consider the following example:

Click here to view code image

```
// stl/print.hpp

#include <iostream>
#include <string>

// PRINT_ELEMENTS()
// - prints optional string optstr followed by
// - all elements of the collection coll
// - in one line, separated by spaces
template <typename T>
inline void PRINT_ELEMENTS (const T& coll,
                            const std::string& optstr="")
{
    std::cout << optstr;
    for (const auto&  elem : coll) {
        std::cout << elem << ' ';
    }
    std::cout << std::endl;
}
```

This example defines a generic function that prints an optional string followed by all elements of the passed container.

Before C++11, the loop over the elements had to look as follows:

```
typename T::const_iterator pos;
for (pos=coll.begin(); pos!=coll.end(); ++pos) {
    std::cout << *pos << ' ';
}
```

Here, `pos` is declared as having the iterator type of the passed container type. Note that `typename` is necessary here to specify that `const_iterator` is a type and not a static member of type `T` (see the introduction of `typename` in Section 3.2, page 34).

In addition to `iterator` and `const_iterator`, containers provide other types to facilitate the writing of generic functions. For example, they provide the type of the elements to enable the handling of temporary copies of elements. See Section 9.5.1, page 468, for details.

The optional second argument of `PRINT_ELEMENTS` is a string that is used as a prefix before all elements are written. Thus, by using `PRINT_ELEMENTS()`, you could comment or introduce the output like this:

```
PRINT_ELEMENTS (coll, "all elements: ");
```

I introduced this function here because I use it often in the rest of the book to print all elements of containers by using a simple call.