Namespaces and the ANSI/ISO C++ Standard Library

The **namespace** mechanism creates a collection of classes, free functions, constant declarations and so forth that are qualified by a name. The syntax for a namespace declaration is:

```
namespace Name
{
   declaration and definitions of classes, functions,
   constants, etc.
}
```

In a program, the elements of a namespace are accessed by using the "`::`" operator. For instance, the namespace MathConstants contains the definitions of two mathematical constants π and e. Access their value in the namespace MathConstants using the syntax `MathConstants::PI and MathConstants::E`.

```
namespace MathConstants
{
    const double PI = 3.141592653589793;
    const double E = 2.718281828459045;
}
```

Namespaces permit the programmer to bundle a variety of C++ constructs in a named collection.

PROGRAM Namespaces-1 DECLARING AND USING NAMESPACES

This short program uses the namespace MathConstants and accesses the constants in the function main(). Assume that the namespace declaration for MathConstants is stored in the header file "mconst.h".

```
#include <iostream>

#include "mconst.h"

using namespace std;

int main()
{
   cout << setreal(1,8) << MathConstants::PI << endl
        << setreal(1,10) << MathConstants::E << endl;

   return 0;
}
```

```
Run:

3.14159265
2.7182818285
```

Duplicate Names

Two namespaces may contain elements with the same name. This can occur when you install two different software packages that support classes, functions, or constant declarations that use the same names. For instance, consider the two namespaces MathConstants and ShortConstants. The two namespaces provide constants of the same name but with different precision.

```
namespace MathConstants
{
    const double PI = 3.141592653589793;
    const double E = 2.718281828459045;
}

namespace ShortConstants
{
    const double PI = 3.14159;
    const double E = 2.71828;
    const double ROOT2 = 1.41421;
}
```

A program references PI in MathConstants using the expression `MathConstants::PI` and PI in ShortConstants using the expression `ShortConstants::PI`.

The Keyword using

The keyword *using* allows the elements of a single namespace to be referenced without the "`::`" operator. All elements of any other namespace must use the "`::`" operator. The syntax for the keyword is

```
using namespace Name;
```

If the code provides the statement

```
using namespace MathConstants;
```

a program statement

```
cout << E << "  " << ShortConstants::E
     << ShortConstants::ROOT2 << endl;
```

outputs the value of E from MathConstants. The references to a constants E and ROOT2 in the namespace ShortConstants must use the "`::`" operator.

The C++ Standard Library

ANSI (American National Standards Institute) and ISO (International Standards Organization) cooperate to develop a worldwide standard for the C++ language called the ANSI/ISO C++ Standard. This standard defines a C++ library that includes the Standard Template Library (STL) of container classes and the string class. The library bundles the components in a namespace called *std*. The C++ Standard library provides 32 C++ header files, as shown in Table Namespaces1.

Table Namespaces1 *C++ Library Header Files*

<algorithm>	<iomanip>	<list>	<queue>	<typeinfo>
<bitset>	<ios>	<locale>	<set>	<utility>
<complex>	<iosfwd>	<map>	<sstream>	<valarray>
<deque>	<iostream>	<memory>	<stack>	<vector>
<exception>	<istream>	<new>	<stdexcept>	
<fstream>	<iterator>	<numeric>	<streambuf>	
<functional>	<limits>	<ostream>	<string>	

The Standard Library also includes 18 additional header files that describe the key facilities of the original library supplied by the C programming language. The header files are provided in Table Namespaces2:

Table Namespaces2 *C++ Header Files for C Library Facilities*

<cassert>	<ciso646>	<csetjmp>	<cstdio>	<cwchar>
<cctype>	<climits>	<csignal>	<cstdlib>	<cwctype>
<cerrno>	<clocale>	<cstdarg>	<cstring>	
<cfloat>	<cmath>	<cstddef>	<ctime>	

To use the components of the standard C++ library, include the necessary header files and add the statement

```
using namespace std;
```

to the program. For instance, to use the I/O library and the string class, include the following statements:

```
#include <iostream>     // accesses I/O stream classes
#include <string>       // string class

using namespace std;    // reference components without using std::
```

PROGRAM ILLUSTRATING NAMESPACE STD

This program declares the C++ string object *str*, inputs its value using the C++ stream I/O library, and outputs the result.

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main()
{
   string first, last, fullname;
   double x;

   cout << "Enter a first and last name:  ";
   // operations from the ANSI string class
   cin >> first >> last;
   fullname = last + ", " + first;
   cout << "The full name is " << fullname << endl;
```

```
   cout << "Enter a number: ";
   cin >> x;
   // sqrt() comes from <cmath>
   cout << "The square root of " << x << " is "
        << sqrt(x) << endl;

   return 0;
}
```

```
Run:

Enter a first and last name:  Standard Ansi
The full name is Ansi, Standard
Enter a number: 5
The square root of 5 is 2.23607
```