When you use different library modules, there is always a potential for name crashes, as different library may use the same name for different purposes. This problem can be resolved via the use of *namespace* in C++. A *namespace* is a collection for identifiers under the same naming scope. (It is known as *package* in UML and Java.) The entity name under a namespace is *qualified* by the namespace name, followed by :: (known as scope resolution operator), in the form of *namespace*::*entityName*.

To place an entity under a namespace, use keyword namespace as follow:

// create a namespace called myNamespace for the enclosed entities

namespace **myNameSpace** {

int foo; // variable

int f() { ...... }; // function

class Bar { ...... }; // compound type such as class and struct

}

// To reference the entities, use

**myNameSpace::**foo

**myNameSpace::**f()

**myNameSpace::**Bar

A namespace can contain variables, functions, arrays, and compound types such as classes and structures.

**Namespace Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <iostream>    namespace a { // contains variables  int i1 = 8;  int i2 = 9;  }    namespace b { // contains function  int max(int n1, int n2) {  return (n1 > n2) ? n1 : n2;  }  }    int main() {  std::cout << a::i1 << std::endl; // 8  std::cout << b::max(a::i1, a::i2) << std::endl; // 9  } |

Namespaces are opened. In other words, you can add more names into an existing namespace using additional namespace definition.

**Using Namespace**

For example, all the identifiers in the C++ standard libraries (such as cout, endl and string) are placed under the namespace called std. To reference an identifier under a namespace, you have three options:

1. Use the fully qualified names, such as std::cout, std::endl, std::setw() and std::string. For example,
2. std::cout << std::setw(6) << 1234 << std::endl;

Missing the "std::" results in "error: 'xxx' was not declared in this scope".

1. Use a *using declaration* to declare the particular identifiers. For example,
2. using std::cout;
3. using std::endl;
4. ......
5. cout << std::setw(6) << 1234 << endl;

You can omit the "std::" for cout and endl, but you still have to use "std::" for setw.

1. Use a *using namespace directive*. For example,
2. using namespace std;
3. ......
4. cout << setw(6) << 1234 << endl;

The using namespace directive effectively brings all the identifiers from the specified namespace to the global scope, as if they are available globally. You can reference them without the scope resolution operator. Take note that the using namespace directive may result in name crashes with identifier in the global scope.

1. For long namespace name, you could define a shorthand (or alias) to the namespace, as follows:

namespace *shorthand* = *namespace-name*;

You can now refer to your class as *shorthand*::*entityName*.

As mentioned, all the standard C++ library components are packaged inside a namespace called std. They include cin, cout, endl, setw(), setprecision(), and string. Hence, we always included a "using namespace std;" directive after the "#include <iostream>" and "#include <string>". You could also use the qualified name such as std::cin, std::cout, std::endl, std::string, instead of the using directive. You could also use selective using such as "using std::cin;", "using std::cout;", "using std::endl;" and "using std::string;".

Compare the following two code samples:

#include <iostream>

using namespace std; // std namespace is available to ALL

int main() { ....... }

#include <iostream>

int main() {

using namespace std; // std namespace is available to main() only

.......

}

Note: you could use the C-style header "iostream.h" which does not use namespace to replace the first two lines in the first sample.

**Global Namespace**

In C++, an entity (variable, function, or class) belongs to the *global namespace* (identified by :: with no namespace name), if it is not enclose within a namespace declaration. For example, main() can be identified as ::main().