

# **Namespaces**

Namespaces allow to group entities like classes, objects and functions under a name. This way the global scope can be divided in "sub-scopes", each one with its own name.

The format of namespaces is:

```
namespace identifier
entities
```

Where identifier is any valid identifier and entities is the set of classes, objects and functions that are included within the namespace. For example:

```
1 namespace myNamespace
2 {
3
   int a, b;
4 }
```

In this case, the variables a and b are normal variables declared within a namespace called myNamespace. In order to access these variables from outside the myNamespace namespace we have to use the scope operator ::. For example, to access the previous variables from outside myNamespace we can write:

```
1 myNamespace::a
2 myNamespace::b
```

The functionality of namespaces is especially useful in the case that there is a possibility that a global object or function uses the same identifier as another one causing redefinition errors. For example

# **Numerical Bases** C++ Language Tutorial Introduction: Instructions for use Basics of C++: Structure of a program Variables. Data Types. Constants **Operators** Basic Input/Output Control Structures: Functions (I) Functions (II) Compound Data Types: Arrays **Character Sequences Pointers** Dynamic Memory **Data Structures** Other Data Types **Object Oriented Programming:** Classes (I) Classes (II)

C++ Language Tutorial

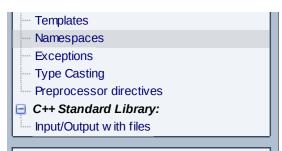
**Boolean Operations** 

Ascii Codes

**Advanced Concepts:** 

Friendship and inheritance

Polymorphism



#### **ICSNetwork**

www.ics.com/icsnetwork

Free Qt Training Videos - Learn New
Techniques for Developing AdChoices

runction uses the same luchtiner as another one, causing reachinition errors, nor example,

```
1 // namespaces
                                                         5
 2 #include <iostream>
                                                         3.1416
 3 using namespace std;
5 namespace first
6 {
7
    int var = 5;
8 }
 9
10 namespace second
11 {
12
    double var = 3.1416;
13 }
14
15 int main () {
16
    cout << first::var << endl;</pre>
17
    cout << second::var << endl;</pre>
18 return 0;
19 }
```

In this case, there are two global variables with the same name: var. One is defined within the namespace first and the other one in second. No redefinition errors happen thanks to namespaces.

#### using

The keyword using is used to introduce a name from a namespace into the current declarative region. For example:

```
1 // using
 2 #include <iostream>
                                                       2.7183
 3 using namespace std;
                                                       10
                                                       3.1416
 5 namespace first
 6 {
    int x = 5;
    int y = 10;
 9 }
10
11 namespace second
12 {
13
    double x = 3.1416;
    double y = 2.7183;
14
15 }
```

```
16
17 int main () {
    using first::x;
using second::y;
    cout << x << endl;
   cout << y << endl;
22
    cout << first::y << endl;</pre>
   cout << second::x << endl;
24
    return 0;
25 }
```

Notice how in this code, x (without any name qualifier) refers to first::x whereas y refers to second::y, exactly as our using declarations have specified. We still have access to first::y and second::x using their fully qualified names.

The keyword using can also be used as a directive to introduce an entire namespace:

```
1 // using
                                                        5
 2 #include <iostream>
                                                        10
 3 using namespace std;
                                                        3.1416
                                                        2.7183
 5 namespace first
 6 {
    int x = 5;
 8
    int y = 10;
9 }
10
11 namespace second
12 {
13
    double x = 3.1416;
14
    double y = 2.7183;
15 }
16
17 int main () {
18
  using namespace first;
19 cout << x << end1;
20 cout << y << endl;
   cout << second::x << endl;</pre>
22
    cout << second::y << endl;</pre>
23
    return 0;
24 }
```

In this case, since we have declared that we were using namespace first, all direct uses of x and y without name qualifiers were referring to their declarations in namespace first.

using and using namespace have validity only in the same block in which they are stated or in the entire code if they are used directly in the global scope. For example, if we had the intention to first use the objects of one namespace and then those of another one, we could do something like:

```
1 // using namespace example
 2 #include <iostream>
                                                          3.1416
 3 using namespace std;
 5 namespace first
 6 {
    int x = 5;
 8 }
 9
10 namespace second
11 {
12
     double x = 3.1416;
13 }
14
15 int main () {
16
17
       using namespace first;
18
       cout << x << endl;</pre>
19
20
21
       using namespace second;
22
       cout << x << endl;</pre>
23
24
     return 0;
25 }
```

### Namespace alias

We can declare alternate names for existing namespaces according to the following format:

namespace new\_name = current\_name;

## Namespace std

All the files in the C++ standard library declare all of its entities within the std namespace. That is why we have generally included the using namespace std; statement in all programs that used any entity defined in iostream.



Home page | Privacy policy © cplusplus.com, 2000-2013 - All rights reserved - v3.1 Spotted an error? contact us