**Namespaces.**

Only one entity can exist with a particular name in a particular scope. This is seldom a problem for local names, since blocks tend to be relatively short, and names have particular purposes within them, such as naming a counter variable, an argument, etc...

But non-local names bring more possibilities for name collision, especially considering that libraries may declare many functions, types, and variables, neither of them local in nature, and some of them very generic.

Namespaces allow us to group named entities that otherwise would have *global scope* into narrower scopes, giving them *namespace scope*. This allows organizing the elements of programs into different logical scopes referred to by names.

The syntax to declare a namespace is:

|  |
| --- |
| namespace identifier  {  named\_entities  } |

Where identifier is any valid identifier and *named\_entities*is the set of variables, types and functions that are included within the *namespace*. For example:

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

In this case, the variables *a* and *b* are normal variables declared within a namespace called *myNamespace*. These variables can be accessed from within their namespace normally, with their identifier (either *a* or *b*), but if accessed from outside the *myNamespace* namespace they have to be properly qualified with the scope operator (*::*).

For example, to access the previous variables from outside *myNamespace* they should be qualified like:

|  |  |  |
| --- | --- | --- |
| 1 2 | myNamespace::a  myNamespace::b |  |

Regular problems when we do not use namespace:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7  8 | // A program to demonstrate need of namespace  int main()  {  int value;  value = 0;  int value = 5; // Error here  double value = 8.1; // Error here  } | Compiler Error: redefinition |  |

Namespaces are particularly useful to avoid name collisions. For example:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21  22  23  24  25  26  27  28 | // namespaces  #include <iostream>  using namespace std;  namespace foo  {  int value()  {  return 5;  }  }  namespace bar  {  const double pi = 3.1416;  double value()  {  return 2\*pi;  }  }  int main ()  {  cout << foo::value() << '\n';  cout << bar::value() << '\n';  cout << bar::pi << '\n';  return 0;  } | 5  6.2832  3.1416 | [Edit & Run](https://cplusplus.com/doc/tutorial/namespaces/) |

In this case, there are two functions with the same name: *value*. One is defined within the namespace *foo*, and the other one in *bar*. No redefinition errors happen thanks to namespaces. Notice also how *pi*is accessed in an unqualified manner from within namespace *bar*(just as *pi*), while it is again accessed in *main*, but here it needs to be qualified as *bar::pi*.

Namespaces can be split: Two segments of a code can be declared in the same *namespace*:

|  |  |  |
| --- | --- | --- |
| 1 2 3 | namespace foo { int a; }  namespace bar { int b; }  namespace foo { int c; } |  |

This declares three variables: *a* and *c* are in namespace *foo*, while *b* is in namespace *bar*.

*Namespaces* can even extend across different translation units (i.e., across different files of source code).

Explanations with examples:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7  8 | // A program to demonstrate need of namespace  int main()  {  int value;  value = 0;  double value; // Error here  value = 0.0;  } | Compiler Error:  'value' has a previous declaration as 'int value' |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | // Here we can see that more than one variable  // are being used without reporting any error.  // That is because they are declared in the  // different namespaces and scopes.  #include <iostream>  using namespace std;  // Variable created inside namespace  namespace first  {  int val = 500;  }  // Global variable  int val = 100;  int main()  {  // Local variable  int val = 200;  // These variables can be accessed from  // outside the namespace using the scope  // operator ::  cout << first::val << '\n';  return 0;  } | // Creating namespaces  #include <iostream>  using namespace std;  namespace ns1  {  int value() { return 5; }  }  namespace ns2  {  const double x = 100;  double value() { return 2 \* x; }  }  int main()  {  // Access value function within ns1  cout << ns1::value() << '\n';  // Access value function within ns2  cout << ns2::value() << '\n';  // Access variable x directly  cout << ns2::x << '\n';  return 0;  } |  |
|  | Output:  500 | Output:  5  200  100 |  |