**using.**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | //using  #include <iostream>  using namespace std;  namespace first  {  int x = 5;  int y = 10;  }  namespace second  {  double x = 3.1416;  double y = 2.7183;  }  int main () {  using first::x;  using second::y;  cout << x << '\n';  cout << y << '\n';  cout << first::y << '\n';  cout << second::x << '\n';  return 0;  } | 5  2.7183  10  3.1416 | [Edit & Run](https://cplusplus.com/doc/tutorial/namespaces/) |

The keyword *using* introduces a name into the current declarative region (such as a block), thus avoiding the need to qualify the name. For example:

Notice how in *main*, the variable *x* (without any name qualifier) refers to *first::x*, whereas *y* refers to *second::y*, just as specified by the using declarations. The variables *first::y* and *second::x* can still be accessed, but require fully qualified names.

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

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | // using  #include <iostream>  using namespace std;  namespace first  {  int x = 5;  int y = 10;  }  namespace second  {  double x = 3.1416;  double y = 2.7183;  }  int main () {  using namespace first;  cout << x << '\n';  cout << y << '\n';  cout << second::x << '\n';  cout << second::y << '\n';  return 0;  } | 5  10  3.1416  2.7183 | [Edit & Run](https://cplusplus.com/doc/tutorial/namespaces/) |

In this case, by declaring that we were using *namespace first*, all direct uses of *x* and *y* without name qualifiers were also looked up in *namespace first*.

*using*and*using namespace* have validity only in the same block in which they are stated or in the entire source code file if they are used directly in the global scope. For example, it would be possible to first use the objects of one namespace and then those of another one by splitting the code in different blocks:

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | // using namespace example  #include <iostream>  using namespace std;  namespace first  {  int x = 5;  }  namespace second  {  double x = 3.1416;  }  int main () {  {  using namespace first;  cout << x << '\n';  }  {  using namespace second;  cout << x << '\n';  }  return 0;  } | 5  3.1416 | [Edit & Run](https://cplusplus.com/doc/tutorial/namespaces/) |

Thence, below code won’t run:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | //using namespace example  #include <iostream>  using namespace std;  namespace first  {  int x = 5;  }  namespace second  {  double x = 3.1416;  }  int main()  {  using namespace first;  cout << x << '\n';  using namespace second;  cout << second::x << '\n';  return 0;  } | Error: Ambiguous ‘x’  One *using namespace* per one scope is valid. | [Edit & Run](https://cplusplus.com/doc/tutorial/namespaces/) |

But it will run if it is written without using namespace, just with qualifying name:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | //using namespace example  #include <iostream>  using namespace std;  namespace first  {  int x = 5;  }  namespace second  {  double x = 3.1416;  }  int main()  {  cout << first::x << '\n';  cout << second::x << '\n';  return 0;  } | 5  3.1416 | [Edit & Run](https://cplusplus.com/doc/tutorial/namespaces/) |