C++ namespaces

- New concept introduced in C++
- Declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it
- Used to organize code into logical groups and to prevent name collisions (e.g. when code includes multiple libraries)
- Scope resolution operator (::) can be used to access variables in other scopes
- ***using*** keyword can be used to introduce specific members of a namespace in to the current scope or all members of the namespace in to the scope of common parent
- Namespaces can be nested

C++ namespaces

```
#include<iostream>
using namespace std;
namespace nspace
{
	int x = 11;
	void fun()
	{
		cout << "Hello!" << endl;
	}
}
int main()
{
	//fun(); // Invalid
	nspace::fun();
	//cout << x << endl; // Invalid
	cout << nspace::x << endl;
	return 0;
}
```

C++ namespaces

```
#include<iostream>
using namespace std;
namespace nspace
{
	int x = 11;
	void fun()
	{
		cout << "Hello!" << endl;
	}
}
using nspace::fun; //using declaration
int main()
{
	fun();
	nspace::fun();
	//cout << x << endl; // Invalid
	cout << nspace::x << endl;
	return 0;
}
```

C++ namespaces

```
#include<iostream>
using namespace std;
namespace nspace
{
	int x = 11;
	void fun()
	{
		cout << "Hello!" << endl;
	}
}
using namespace nspace; //using directive
int main()
{
	fun();
	nspace::fun();
	cout << x << endl;
	cout << nspace::x << endl;
	return 0;
}
```

C++ namespaces

```
#include<iostream>
using namespace std;
int x = 10;
namespace nspace
{
	int x = 20;
	void fun()
	{
		int x = 30;
		cout << x << " " << nspace::x << " " << ::x << endl;
	}
}
int main()
{
	nspace::fun();
	return 0;
}
```

C++ namespaces

- A namespace can be split into multiple blocks and those blocks can be in the same file or separate files.
- One file can contain blocks of multiple namespaces

1.cpp

```
namespace abc
{
        ...
}

namespace xyz
{
        ...
}

namespace abc
{
        ...
}
```

2.cpp

```
namespace xyz
{
        ...
}

namespace abc
{
        ...
}

namespace xyz
{
        ...
}
```

C++ namespaces (using declaration)

```
#include<iostream>
int x = 1;
namespace nspace
{
	int x = 2;
	int y = 3;
}
int main()
{
	int y = 4;
	//error: ‘y’ is already declared in this scope
	//using nspace::y;
	using nspace::x;
	std::cout << x << " " << y;
	int fun();
	fun();
	return 0;
}
int fun()
{
	std::cout << x;
}
```

- Here **nspace::x** has been introduced within main function scope

C++ namespaces (using declaration)

```
#include<iostream>

namespace test {
    int x = 7;
}

namespace test2 {
    using test::x;
}

int main() {
    std::cout << test2::x;
    return 0;
}
```

- **Outpu**
  t
- 7

C++ namespaces (using directive)

```
#include<iostream>

int x = 1;

namespace nspace
{
    int x = 2;
    int y = 3;
    int z = 4;
}
```

- Qualified name lookup for ::x, ::y and ::z, starts from global scope, and only ::x is considered as part of global scope. naspace::y and naspace::z are not considered part of global scope for qualified name lookup.

```
int main()
{
    int y = 5;
    using namespace nspace;
    //error: reference to ‘x’ is ambiguous
    //std::cout << " " << x;
    std::cout << " " << ::x;
    std::cout << " " << y;
    //error: ‘::y’ has not been declared
    //std::cout << " " << ::y;
    std::cout << " " << z;
    //error: ‘::z’ has not been declared
    //std::cout << " " << ::z;
    int fun();
    fun();
    return 0;
}
void fun()
{
    std::cout << " " <<  x;
}
```

- **nspace::x** is qualified name of **x**.
- **Unqualified name** is a name that does not appear to the right of a scope resolution operator ::
- x is unqualified name of variable x.

- For **unqualified name lookup** form **main** function (**after using directive statement**) it is as if all members of the namespace nspace are part of **global scope**
- Produces error **only if** we **try to access x** within main function (**after using directive statement**)
- Unqualified name lookup for x starts from main (local scope), but x is not present in main hence it checks global scope where it find more than one declaration of x hence the error
- Unqualified name lookup for y starts from main (local scope), and y is present in main hence search stops there and it uses local y from main
- Unqualified name lookup for z starts from main (local scope), but z is not present in main hence it checks global scope where it find one declaration of z.

C++ namespaces (using directive)

```
#include<iostream>

int x = 1;

namespace nspace
{
    int x = 2;
    int y = 3;
    int z = 4;
}
```

```
using namespace nspace;
int main()
{
    int y = 5;
    //error: reference to ‘x’ is ambiguous
    //std::cout << " " << x;
    std::cout << " " << ::x;
    std::cout << " " << y;
    std::cout << " " << ::y;
    std::cout << " " << z;
    std::cout << " " << ::z;
    int fun();
    fun();
    return 0;
}
void fun()
{
    //error: reference to ‘x’ is ambiguous
    //std::cout << " " <<  x;
}
```

- Avoid **using namespace std;** as it pollutes the global scope for unqualified name lookup.

- Qualified name lookup for ::x, ::y and ::z, starts from global scope, and only ::x is considered as part of global scope. naspace::y and naspace::z are not considered part of global scope for qualified name lookup.
- In this case using directive statement is present in global scope. **So when y and z are not found in global scope, search continues within namespaces for which using directive is present in global scope (nspace in this case).** And hence search for ::y and ::z ends with nspace::y and nspace::z respectively.

- As using directive statement is present in global scope now, for **unqualified name lookup after using directive statement,** it is as if all members of the namespace nspace are part of **global scope**
- Hence unqualified name lookup for **x** from function **fun** results in ambiguity.

C++ namespaces (using directive)

```
#include<iostream>

namespace test {
    int x = 7;
}

namespace test2 {
    using namespace test;
}

int main() {
    std::cout << test2::x;
    return 0;
}
```

- **Outpu**
  t
- 7

- Qualified name lookup for test2::x starts from test2. So when x is not found in test2, search continues within namespaces for which using directive is present in test2 scope (test namespace in this case). And hence search for test2::x ends with test::x.

C++ namespaces (using directive)

```
#include<iostream>
int x = 1;
namespace test {
    namespace test2 {
        int x = 2;
    }
    namespace test3 {
        using namespace test2;
        void fun() {
            std::cout << x;
            //error: ‘x’ is not a member of ‘test’
            //std::cout << test::x;
            std::cout << " " << ::x;
        }
    }
}
int main() {
    test::test3::fun();
    return 0;
}
```

- **Outpu**
  t
- 2 1

- Qualified name lookup for test::x starts from test namespace. So when x is not found in test namespace, search continues within namespaces for which using directive is present in test namespace (there is no using directive directly within test). Hence search stops there without finding x, and hence compiler generates an error.

- Qualified name lookup for ::x starts directly from global scope and x is present in global scope.

- For unqualified lookup for x, it searches in local scope (fun function scope) first. Variable x is not present in fun function and hence it will look into parent scope of fun scope (test3). x is not present in test3. Hence search continues to its parent (test). Variable x is found in test. Variable x is considered part of test (for unqualified name lookup) because of using directive in test3.