Namespace in C++ 👍

A namespace is a container for the identifier. A namespace defines a scope where identifiers like variables, functions, classes, etc., are declared. The main purpose of using a namespace is to prevent ambiguity when two identifiers have the same name.

What is Namespace?

A namespace in C++ is a declarative region that provides a scope to the identifiers (names of types, function, variables etc.) inside it. It is used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

Syntax of namespace

In C++, a namespace is defined using the syntax namespace **namespace_name { /* declarations */ }.** We can define variables, functions, and other named entities inside the namespace. To access the named entities from outside the namespace, we use the **namespace_name:: prefix.**
In C++, the syntax of the namespace is the keyword itself. You can declare a class by using the keyword as namespace. You can also use the declaration for a single identifier as :

```
using std:: string
```

or, in the namespace, for all the identifiers by keyword **using.**

```
using namespace std;
```

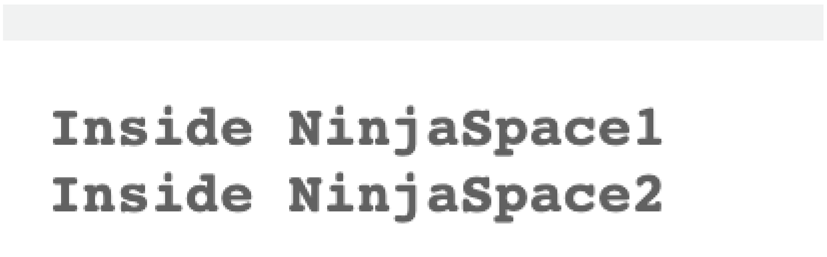
What is Nested Namespace?

```
namespace NinjaSpace1
 {
   // code declarations
    namespace NinjaSpace2
    {
      // code declarations
    }
 }
```

The above code shows how namespaces can be nested. To access **NinjaSpace2**, we have to write: ***‘using namespace NinjaSpace1::NinjaSpace2’*** and to access **NinjaSpace1**: ***‘using namespace NinjaSpace1’.***
By using space1, the elements of space2 will also be available in the scope as well. For example:
Code

```
#include <iostream>
using namespace std;


//  First name space
namespace NinjaSpace1{
    void Function(){
     cout << "Inside NinjaSpace1" << endl;
    }

  // second name space
  namespace NinjaSpace2{
    void Function(){
       cout << "Inside NinjaSpace2" << endl;
   }
 }

}

using namespace NinjaSpace1;

int main () {

// Calling the function from NinjaSpace1
Function();

// Calling the function from NinjaSpace2
NinjaSpace1::NinjaSpace2::Function();

return 0;
}
```

Output



```
Inside NinjaSpace1
Inside NinjaSpace2
```

In the above code, we have used the line: ***using namespace NinjaSpace1;*** which instructs the compiler we are using the methods defined within the Ninjaspace1. Moving on, the line, ***NinjaSpace1::NinjaSpace2::Function();*** instructs the compiler that we are calling the function defined within NinjaSpace2;

How can we use a Namespace?

```
#include <iostream>
using namespace std;
int main()
{
  int k;
  k=0;
  float k;
  k=0.0;
  cout<<k<<endl;
  return 0;
}
```

In the above code, we will get a compilation error because variable k is previously declared as int k. Two variables with the same name cannot be used in the same scope.
We can efficiently resolve the above problem by using namespace.
We use namespaces to avoid name collisions.
For example:

```
// namespaces
#include <iostream>
using namespace std;
namespace space1
```

```
{
  int val() { return 5; }
}
namespace space2
{
  double x = 10.4;
  double val() { return 2*x; }
}
int main () {
  cout << space1::val() << endl;
  cout << space2::val() << endl;
  cout << space2::x << endl;
  return 0;
}
```

Output:

```
5
20.8
10.4
```

In the above code, two functions are with the same name: “val”. One is defined within the Namespace space1, and the other one in Namespace space2. So there are no redefinition errors that will happen thanks to the namespace. Notice also how x is accessed in an unqualified way from within namespace space2, while it is again accessed in the main function, it needs to be qualified as space2::x.
Namespaces can be split into two segments of a code that can be declared in the same namespace:

1. namespace space1 { int a; }
2. namespace space1 { int b; }
3. namespace space2 { int c; }

This declares three variables: a and b are in namespace space1, while b is in namespace space2.
You can also do a free certification of Basics of C++.

Declaring namespaces and namespace members

In general, you declare the namespace in the header file so that all your directives and functions can be easily used. But if in case, you have class in different file, you can explicitly use the namespace as below:
appdata.cpp

```
namespace Application
{
void show()
}
```

When you use this class in a different class, using a directive, you must use the fully qualified name as shown below:
main.cpp

```
#include “appdata.h
using namespace appdata;
void show::print()
{
court<<"Data displayed:"
}
```

A namespace can be declared in multiple files, and when these files are compiled, all the classes and functions are loaded at once. Members of the namespace can be declared in a different file by explicitly qualifying the name of the function. However, you must take care that the declaration of the method or member is defined after the namespace declaration is done.

The global namespace

When in the explicit namespace you did not declare any identifier, it is considered to be a part of the global namespace. It is generally suggested that you must not declare any functions or methods as global. When you declare the namespace in the global scope, the entire program is referred to that global scope. However, declaring variables and functions at the global level is not promoted, as it disrupts the principle of data security. Whenever you want to explicitly declare a namespace other than the global level, use the resolution operator (::).

The std namespace

The std namespace plays a vital role in C++. In the std namespace, all the C++ standard libraries' types and functions are declared. In C++, all the functionalities are declared inside the standard library. The basic operations like input/output, containers, algorithms, and function declaration are all well-defined in the std namespace. It helps to avoid name conflicts of the functions.

Defining a Namespace

You can define a namespace in C++ using the keyword ***'namespace'***. For example:

```
namespace NinjaNamespace{
   // Code

   // Declaring variables and functions
   int NinjaVar1;

}
```

To access the variables and methods defined within the 'NInjaNamespace', you need to access it like this:

```
NinjaNamespace :: NinjaVar1 ;
```

Another method is to use a ***namespace directive*** to access the variables and methods defined within the namespace. The directive instructs the compiler that the upcoming code uses the names defined in the specific namespace.

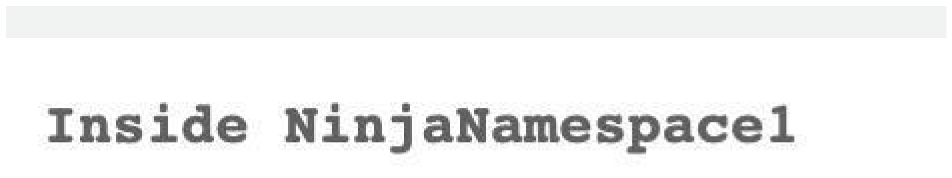
Example

Implementation

```
#include <iostream>
using namespace std;

// Namespace 1
namespace NinjaNamespace1 {
  void Function() {
     cout << "Inside NinjaNamespace1" << endl;
   }
}

// second name space
```

```
namespace NinjaNamespace2{
   void Function() {
      cout << "Inside NinjaNamespace2" << endl;
   }
}

using namespace NinjaNamespace1;

int main (){

// Calling the function from NinjaNamespace2
Function();

return 0;
}
```

Output



```
Inside NinjaNamespace1
```

The line ***'using namespace NinjaNamespace1;'*** instructs the compiler that we are using the variables defined in ***NinjaNamespace1***.

Advantage of Namespace to avoid name collision.

- Namespace in C++ is used to avoid naming collisions.

- Naming collision is a situation when the same variable is assigned to two or more different entities in a given namespace or scope.

- For example, assume you have written a program that consists of a function ***function1()***, and there is another library that contains the function with the same name ***function1()***. Here, the compiler doesn't know which ***function1()*** we are referring to. So, this problem can be solved using a ***namespace*** in