Read Discuss Courses Practice Video

Namespace in C++ | Set 1 (Introduction)

Difficulty Level : Medium • Last Updated : 27 Jan, 2023

- Namespace provide the space where we can define or declare identifier i.e. variable, method, classes.
- Using namespace, you can define the space or context in which identifiers are defined i.e. variable, method, classes. In essence, a namespace defines a scope.

Advantage of Namespace to avoid name collision.

- Example, you might be writing some code that has a function called xyz() and there is another library available which is also having same function xyz(). Now the compiler has no way of knowing which version of xyz() function you are referring to within your code.
- A namespace is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries.
- The best example of namespace scope is the C++ standard library (std) where all the classes, methods and templates are declared. Hence while writing a C++ program we usually include the directive using namespace std;

Defining a Namespace:

- A namespace definition begins with the keyword namespace followed by the namespace name as follows:

Start Your Coding Journey Now! Login Register

```
namespace  namespace_name
```

Read Discuss Courses Practice Video

```
    method (void add();)
    classes ( class student{};)
}
```

- It is to be noted that, there is no semicolon (;) after the closing brace.
- To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:
- namespace_name: :code; // code could be variable , function or class.

The using directive:

- You can also avoid prepending of namespaces with the using namespace directive. This directive tells the compiler that the subsequent code is making use of names in the specified namespace.
- The namespace is thus implied for the following code:

C++

```
#include <iostream>
using namespace std;
// first name space
namespace first_space
{
  void func()
  {
     cout << "Inside first_space" << endl;
  }
}

// second name space
namespace second_space
{
  void func()
  {
     cout << "Inside second_space" << endl;
```

Start Your Coding Journey Now!

```
int main ()
{
    return 0;
}
```

Read Discuss Courses Practice Video

Output

```
Inside first_space
```

- Names introduced in a using directive obey normal scope rules. The name is visible found. Entities with the same name defined in an outer scope are hidden.

Data Structures and Algorithms Interview Preparation Data Science Topic-wise Practice C

Nested Namespaces:

- Namespaces can be nested where you can define one namespace inside another name space as follows:

```
SYNTAX:
    namespace namespace_name1
    {
          // code declarations
          namespace namespace_name2
          {
             // code declarations
          }
    }
```

You can access members of nested namespace by using resolution operators as follows:
// to access members of namespace_name2
using namespace namespace_name1::namespace_name2;
// to access members of namespace_name1
using namespace namespace_name1;

Start Your Coding Journey Now!

In the above statements if you are using namespace_name1, then it will make elements

Read Discuss Courses Practice Video

C++

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

// first name space
namespace first_space
{
  void func()
  {
     cout << "Inside first_space" << endl;
  }
  // second name space
  namespace second_space
  {
     void func()
     {
        cout << "Inside second_space" << endl;
     }
  }
}
using namespace first_space::second_space;
int main ()
{
    // This calls function from second name space.
      func();

      return 0;
}
```

Output

```
Inside second_space
```

Let us see how namespace scope the entities including variable and functions:

C++

Start Your Coding Journey Now!

```
using namespace std;
// first name space
```

Read Discuss Courses Practice Video

```
  void func()
  {
     cout << "Inside first_space" << endl;
  }
}
// second name space
namespace second_space
{
  void func()
  {
     cout << "Inside second_space" << endl;
  }
}

int main ()
{
       // Calls function from first name space.
      first_space :: func();
    // Calls function from second name space.
      second_space :: func();
      return 0;
}
```

Output

```
Inside first_space
Inside second_space
```

Consider the following C++ program:

CPP

Read Discuss Courses Practice Video

```
int main()
{
    int value;
    value = 0;
    double value; // Error here
    value = 0.0;
}
```

Output :

```
Compiler Error:
'value' has a previous declaration as 'int value'
```

In each scope, a name can only represent one entity. So, there cannot be two variables with the same name in the same scope. Using namespaces, we can create two variables or member functions having the same name.

CPP

```
// Here we can see that more than one variables
// 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
```

Start Your Coding Journey Now!

```
    // outside the namespace using the scope
    // operator ::
```

Read Discuss Courses Practice Video

```
    return 0;
}
```

Output

```
500
```

Definition and Creation: 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. Namespaces provide the space where we can define or declare identifiers i.e. names of variables, methods, classes, etc.

- A namespace is a feature added in C++ and is not present in C.
- A namespace is a declarative region that provides a scope to the identifiers (names of functions, variables or other user-defined data types) inside it.
- Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope.

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows:

```
namespace namespace_name
{
   int x, y; // code declarations where
             // x and y are declared in
             // namespace_name's scope
}
```

- Namespace declarations appear only at global scope.
- Namespace declarations can be nested within another namespace.
- Namespace declarations don't have access specifiers (Public or Private).
- No need to give a semicolon after the closing brace of the definition of namespace

Start Your Coding Journey Now!

Defining a Namespace:

A namespace definition begins with the keyword namespace followed by the namespace

Read Discuss Courses Practice Video

C++

```
namespace  namespace_name{
     // code declarations i.e. variable  (int a;)
     method (void add();)
     classes ( class student{};)
}
```

It is to be noted that there is no semicolon (;) after the closing brace.

To call the namespace-enabled version of either a function or a variable, prepend the namespace name as follows:

namespace_name: :code; // code could be a variable, function or class.

C++

```
// Let us see how namespace scope the entities including variable and functions:

#include <iostream>
using namespace std;

// first name space
namespace first_space
{
   void func()
   {
      cout << "Inside first_space" << endl;
   }
}

// second name space
namespace second_space
{
   void func()
   {
```

Start Your Coding Journey Now!

```
int main ()
```

Read Discuss Courses Practice Video

```
   first_space::func();
    // Calls function from second name space.
   second_space::func();
   return 0;
}

// If we compile and run above code, this would produce the following result:
// Inside first_space
// Inside second_space
```

Output

```
Inside first_space
Inside second_space
```

The using directive:

You can avoid prepending of namespaces with the using namespace directive. This directive tells the compiler that the subsequent code is making use of names in the specified namespace. The namespace is thus implied for the following code:

C++

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

// first name space
namespace first_space
{
   void func()
   {
      cout << "Inside first_space" << endl;
   }
}

// second name space
namespace second_space
{
   void func()
```

Start Your Coding Journey Now!

```
}
```

Read Discuss Courses Practice Video

```
int main ()
{
    // This calls function from first name space.
   func();
   return 0;
}

// If we compile and run above code, this would produce the following result:
// Inside first_space
```

Output

```
Inside first_space
```

Instead of accessing the whole namespace, another option (known as *using* declaration) is to access a particular item within a namespace. For example, if the only part of the std namespace that you intend to use is cout, you can refer to it as follows:
using std::cout;
Subsequent code can refer to cout without prepending the namespace, but other items in the std namespace will still need to be explicit as follows:

C++

```
#include <iostream>
using std::cout;

int main ()
{
    cout << "std::endl is used with std!" << std::endl;
      return 0;
}
```

Output

```
std::endl is used with std!
```

Start Your Coding Journey Now!

Names introduced in a using directive obey normal scope rules, i.e., they are visible from

C++

```
// Creating namespaces
#include <iostream>
using namespace std;
namespace ns1 {
int value() { return 5; }
} // namespace ns1
namespace ns2 {
const double x = 100;
double value() { return 2 * x; }
} // namespace ns2

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:

```
5
200
100
```

Classes and Namespace: The following is a simple way to create classes in a namespace:

Start Your Coding Journey Now!

```
// A C++ program to demonstrate use of class
```

Read Discuss Courses Practice Video

```
using namespace std;

namespace ns
{
    // A Class in a namespace
    class geek
    {
    public:
        void display()
        {
            cout<<"ns::geek::display()"<<endl;;
        }
    };
}

int main()
{
    // Creating Object of geek Class
    ns::geek obj;

    obj.display();

    return 0;
}
```

Output

```
ns::geek::display()
```

A class can also be declared inside namespace and defined outside namespace using the following syntax:

CPP

```
// A C++ program to demonstrate use of class
// in a namespace
#include <iostream>
using namespace std;
```

Start Your Coding Journey Now!

```
} // namespace ns
```

Read Discuss Courses Practice Video

```
public:
    void display() { cout << "ns::geek::display()\n"; }
};

int main()
{
    // Creating Object of geek Class
    ns::geek obj;
    obj.display();
    return 0;
}
```

Output

```
ns::geek::display()
```

We can define methods as well outside the namespace. The following is an example code:

C++

```
// A C++ code to demonstrate that we can define
// methods outside namespace.
#include <iostream>
using namespace std;

// Creating a namespace
namespace ns {
void display();
class geek {
public:
    void display();
};
} // namespace ns

// Defining methods of namespace
void ns::geek::display()
{
    cout << "ns::geek::display()\n";
}
```

Start Your Coding Journey Now!

```
// Driver code
int main()
```

Read Discuss Courses Practice Video

```
    ns::display();
    obj.display();
    return 0;
}
```

Output:

```
ns::display()
ns::geek::display():
```

Nested Namespaces:

Namespaces can be nested, i.e., you can define one namespace inside another namespace as follows:

C++

```
namespace namespace_name1 {
  // code declarations
  namespace namespace_name2 {
     // code declarations
  }
}
```

You can access the members of a nested namespace by using the resolution operator (::) as follows:

C++

```
// to access members of namespace_name2
```

Start Your Coding Journey Now!

```
using namespace namespace_name1;
```

Read Discuss Courses Practice Video

elements of namespace_name2 available in the scope as follows:

C++

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

// first name space
namespace first_space
{
   void func()
   {
      cout << "Inside first_space" << endl;
   }
   // second name space
   namespace second_space
   {
      void func()
      {
         cout << "Inside second_space" << endl;
      }
   }
}
using namespace first_space::second_space;
int main ()
{

   // This calls function from second name space.
   func();

   return 0;
}

// If we compile and run above code, this would produce the following result:
// Inside second_space
```

Output

```
Inside second_space
```

Start Your Coding Journey Now!

For example, you might be writing some code that has a function called xyz() and there

Read Discuss Courses Practice Video

within your code.

A namespace is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables, etc. with the same name available in different libraries.

The best example of namespace scope is the C++ standard library (std), where all the classes, methods and templates are declared. Hence, while writing a C++ program, we usually include the directive

using namespace std;

namespace in C++ | Set 2 (Extending namespace and Unnamed namespace) Namespace in C++ | Set 3 (Accessing, creating header, nesting and aliasing) Can namespaces be nested in C++? **Reference**: http://www.cplusplus.com/doc/tutorial/namespaces/ This article is contributed by **Abhinav Tiwari**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Like 208

Previous Next

Start Your Coding Journey Now!

Read Discuss Courses Practice Video

2. Namespace in C++ | Set 3 (Accessing, creating header, nesting and aliasing)
3. Cons of using the whole namespace in C++
4. Why "using namespace std" is considered bad practice
5. Difference between namespace and class
6. Why it is important to write "using namespace std" in C++ program?
7. How to Run a C++ Program Without Namespace?
8. Forward List in C++ | Set 1 (Introduction and Important Functions)
9. Understanding "volatile" qualifier in C | Set 1 (Introduction)
10. Hygienic Macros : An Introduction

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Start Your Coding Journey Now!

manojqqh5, psych0, vinayedula, qwerty201103, nitinprbil

Read Discuss Courses Practice Video

Practice Tags : CPP

Improve Article Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company
About Us
Careers
In Media
Contact Us
Privacy Policy
Copyright Policy
Advertise with us

Learn
DSA
Algorithms
Data Structures
SDE Cheat Sheet
Machine learning
CS Subjects
Video Tutorials
Courses

News
Top News
Technology
Work & Career
Business

Languages
Python
Java
CPP
Golang

Start Your Coding Journey Now!

Knowledge

SQL

Read Discuss Courses Practice Video

Web Development

Web Tutorials

Django Tutorial

HTML

JavaScript

Bootstrap

ReactJS

NodeJS

Contribute

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship

@geeksforgeeks , Some rights reserved

Start Your Coding Journey Now!