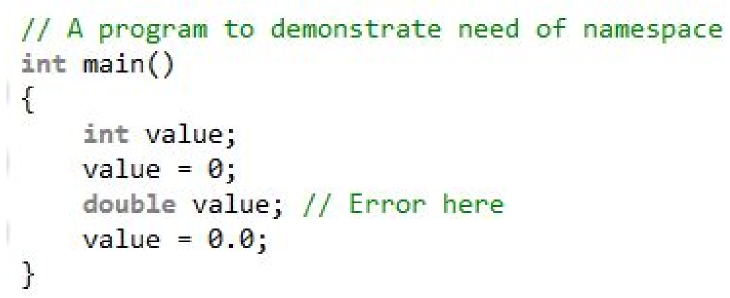
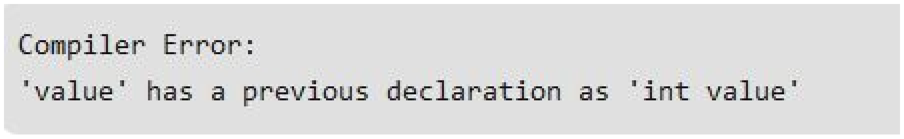
Namespace in C++

Consider following C++ program.



```cpp
// A program to demonstrate need of namespace
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.

Introduction:

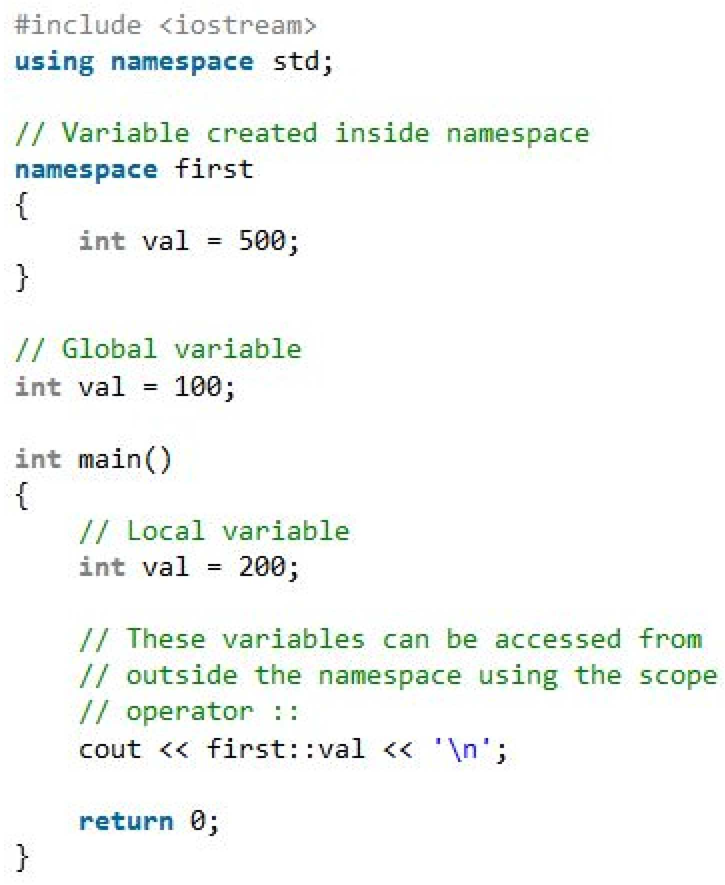
Namespace allows us to group entities under a name. This way the global scope can be divided in "sub-scopes", each one with its own name. The format of namespace is:

```
namespace identifier_name
{
    //entities
}
```

where identifier is any valid identifier and entities can be classes, objects , functions or variables that are included within the namespace. For example:

```
namespace test
{
  int a,b;
}
```

The problem specified above can be solved using namespace . Following is its example:



```
#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;
}
```

Output:

```
500
```

Function and namespace:

Following example shows function within namespace:

```
#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;
   }
}

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
```

Class and namespace:

Following is a simple way to create classes in a name space:

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

namespace ns
{
    // A Class in a namespace
    class geek
    {
    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()
```

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 :

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

   return 0;
}
```

Output:

```
Inside first_space
```