Namespaces in c++

1. What is namespace?

-> Namespace is a declarative region that provides a scope to the identifiers(names of types, functions and variables) inside it.

-> Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

-> Identifiers outside the namespace can access the members by using the fully qualified name for each identifier

-> a. For example std::vector<std::string> vec;,

b. else by a using Declaration for a single identifier (using std::string)

c. using Directive for all the identifiers in the namespace (using namespace std;)

2. How to create namespace?

Namespaces can be created by namespace keyword followed by namespace_name.

```
namespace namespace_name
{
int x,y;
}
```

-> 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 semicolon after the closing brace of definition of namespace.
-> We can split the definition of namespace over several units.

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

-> From the above example we can observe that variable val is used multiple times and still compiler doesnt throw any error.

3. How to access symbols from namespace?

-> Members inside the namespace can be accessed by using namespace name::member name.
Example 1: ns1::value() where ns1 is namespace and value() is a function inside ns1.

Example 2:

```
#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();// Accessing func() symbol from first_Space namespace

   // Calls function from second name space.
   second_space::func(); // Accessing func() from second_space namespace

   return 0;
}
```

4. Usage of "using" keyword

Using keyword is used to:
a. Bring a specific member from the namespace into the current scope.
b. Bring all members from the namespace into the current scope.
c. Bring a base class method or variable into the current class scope.

Examples:
a. *Bringing string and cout in the current scope*

```
#include <iostream>

int main() {
  using std::string;
  using std::cout;
  string s = "Hello World";
  cout << s;
  return 0;
}
```

b. *Bringing the entire std namespace in the current scope*

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

int main() {
  cout << "Hello World";
  return 0;
}
```

c. *Bringing greet() in the scope of Derived class*

```
#include <iostream>
using namespace std;
class Base {
   public:
      void greet() {
         cout << "Hello from Base" << endl;;
      }
};
class Derived : Base {
   public:
      using Base::greet;
      void greet(string s) {
         cout << "Hello from " << s << endl;
         // Instead of recursing, the greet() method
         // of the base class is called.
         greet();
      }
};
int main() {
   Derived D;
```

```
  D.greet("Derived");
  return 0;
}
```

d. *Bringing sum() in the scope of the Derived class*

```
#include <iostream>
using namespace std;
class Base {
  public:
    void sum(int a, int b) {
      cout << "Sum: " << a + b << endl;;
    }
};
class Derived : protected Base {
  public:
    using Base::sum;
};
int main() {
  Derived D;
  // Due to protected inheritance, all the public methods
  // of the base class become protected methods of the
  // derived class. If the using keyword word was not used,
  // calling sum like this would not have been possible.
  D.sum(10, 20);
  return 0;
}
```

5. Multilevel/Nested namespace

Let us understand the nested namespace with an example:

```
#include <iostream>

int x = 20;
namespace outer {
 int x = 10;
 namespace inner {
   int z = x; // this x refers to outer::x
 }
}

int main()
{
 std::cout<<outer::inner::z; //prints 10
 getchar();
 return 0;
}
```

Output: 10

-> Here we can see that namespace inner is created inside namespace outer, which is inside the global namespace.

-> In the line int z = x, x refers to outer ::x. If x would not have been in outer then this x would have referred to x in the global namespace.

6. Default namespace

C++ has a default namespace named std, which contains all the default library of the C++ included using #include directive.

7. Anonymous namespace

Anonymous namespace is a namespace with no name. They are directly usable in the same program and are used for declaring unique identifiers. Also avoids making global static variable. Anonymous namespace is only accessible within the file you created it in.

Example:

File 1.cpp

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

        namespace
        {
                int local;
        }

        void func();

        int main()
        {
                local = 1;
                cout << "Local=" << local << endl;
                func();
                cout << "Local=" << local << endl;
                return 0;
        }
```

File2.cpp

```
        namespace
        {
                // Should not collide with other files
                int local;
        }

        void func()
        {
                local = 2;
        }
```

Note: Include both files in the same project.

-> Result of this program should be:
Local=1
Local=1

-> It should not be
Local=1
Local=2

-> We can observe from the above that scope of unnamed namespace is within the file.

-> The example below demonstrates improper usage of unnamed/anonymous namespace

```
#include <iostream>

using namespace std;

namespace {
  const int i = 4;
}

int i = 2;

int main()
{
  cout << i << endl; // error
  return 0;
}
```

-> Here the variable i is present both as global variable and in unnamed namespace. So compiler cannot distinguish between both.So namespace must be uniquely identified with an identifier i and i must specify the namespace it is using..

8. What is the use of unnamed namespace?

-> Members inside the unnamed namespace can only be accessed in same file.

->Unnamed namespace limits access of class, variables, function and objects to the file which it is defined.

-> Functionality is similar to static keyword. Static variables limits the access of global variable and functions to the file in which they are defined.

-> Difference between unnamed namespace and static keyword is that static keyword can be used with variable, function and objects but not with user defined class.

Example: static int x; //Correct

But,

static class xyz {Body of class}//Wrong
static structure {Body of structure} //Wrong

But same can be possible with unnamed namespace. For example,

```
 namespace {
        class xyz {/*Body of class*/}
        static structure {/*Body of structure*/}
  } //Correct
```

9. Why “using namespace std” is considered bad practice?

-> Alternative to this statement is to specify the namespace to which identifier belongs using the scope resolution operator(::).

-> Although using namespace std saves us time from typing std:: everytime, it imports the entire std namespace into current namespace program.

Examples to demonstrate why using namespace std is bad

Example 1:

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

cout << " Something to Display";
```

-> Assume at later stage of development, we wish to use another version of cout that is custom implemented in some library called "foo".

Example 2:

```
#include <foo.h>
#include <iostream>
using namespace std;

cout << " Something to display";
```

-> Here there is ambiguity which version of cout to point to?
So compiler will not compile and throw error.

-> This has destroyed the purpose of namespace which was introduced to resolve identifier name conflicts. But here instead of resolving a name conflict, we are actually creating naming conflict.