### Namespace

Namespace is a container for identifiers. It puts the names of its members in a distinct space so that they don't conflict with the names in other namespaces or global namespace.

#### Creating a Namespace

Creating a namespace is similar to creation of a class.

namespace MySpace

{

// Declarations

}

int main() {}

This will create a namespace called MySpace, inside which we can put our member declarations.

#### Rules to create Namespace

1. The namespace definition must be done at global scope, or nested inside another namespace.
2. Namespace definition doesn't terminates with a semicolon like in class definition.
3. You can use an alias name for your namespace name, for ease of use.

**Example for Alias :**

namespace StudyTonightDotCom

{

void study();

class Learn { };

}

namespace St = StudyTonightDotCom; // St is now alias for StudyTonightDotCom

1. You cannot create instance of namespace.
2. There can be **unnamed** namespaces too. Unnamed namespace is unique for each translation unit. They act exactly like named namespaces.

**Example for Unnamed namespace :**

namespace

{

class Head { };

class Tail { };

int i,j,k;

}

int main() { }

1. A namespace definition can be continued and extended over multiple files, they are not redefined or overriden.

**Example :**

**Header1.h**

namespace MySpace

{

int x;

void f();

}

**Header2.h**

#include "Header1.h";

namespace MySpace

{

int y;

void g();

}

#### Using a Namespace

There are three ways to use a namespace in program,

1. Scope Resolution
2. The using directive
3. The using declaration

#### With Scope Resolution

Any name (identifier) declared in a namespace can be explicitly specified using the namespace's name and the scope resolution :: operator with the identifier.

namespace MySpace

{

class A

{

static int i;

public:

void f();

};

class B; // class name declaration

void func(); //gobal function declaration

}

int **MySpace::A::i**=9; // Initializing static class variable

class **MySpace::B**

{

int x;

public:

int getdata()

{

cout << x;

}

B(); // Constructor declaration

}

**MySpace::B**::B() // Constructor definition

{

x=0;

}

#### The using directive

**using** keyword allows you to import an entire namespace into your program with a global scope. It can be used to import a namespace into another namespace or any program.

**Namespace1.h**

namespace X

{

int x;

class Check

{

int i;

};

}

**Namespace2.h**

include "Namespace1.h";

namespace Y

{

using namespace X;

Check obj;

int y;

}

We imported the namespace X into namespace Y, hence class Check is available in namespace Y.

**Program.cpp**

#include "Namespace2.h";

void test()

{

using Namespace Y;

Check obj2;

}

Hence, the using directive makes it a lot easier to use namespace, wherever you want.

#### The using declaration

When we use **using directive**, we import all the names in the namespace and they are available throughout the program, that is they have global scope.

But with **using declaration**, we import one specific name at a time which is available only inside the current scope.

**NOTE** : Name imported with **using declaration** can override the name imported with **using directive**

**Namespace.h**

namespace X

{

void f() {}

void g() {}

}

namespace Y

{

void f() {}

void g() {}

}

**Program.cpp**

#include "Namespace.h";

void h()

{

using namespace X; // using directive

using Y::f; // using declaration

f(); // calls f() of Y namespace

X::f(); // class f() of X namespace

}

In using declaration, we never mention the argument list of a function while importing it, hence if a namespace has overloaded function, it will lead to ambiguity.