***Object Oriented Programming***

->Namespace:

#include <iostream>

using namespace std;

namespace ns1

{

    int a, b;

}

namespace ns2 // multiple namespaces may have same variable names

{

    int a, b;

}

int main()

{

    int a = 10, b = 20;

    cout << "a=" << a << "b=" << b<<endl;

    ns1::a = 1; // fully qualified name

    ns1::b = 2;

    cout << "ns1 a = " << ns1::a;

    cout << "ns1 b = " << ns1::b<<endl;

    ns2::a = 1;

    ns2::b = 2;

    cout << "ns2 a = " << ns2::a; // fully qualified name

    cout << "ns2 b = " << ns2::b;

}

->classes under namespaces

#include <iostream>

using namespace std;

namespace ns

{

    int a = 500;

    class Sample

    {

        int a; // member variable of Sample class is visible only within the class definition

    public:

        Sample(int a = 0)

        {

            this->a = a;

        }

        void display()

        {

            cout << "in display sample ns"

                 << "a=" << a << endl;

            cout << "a of ns=" << ns::a << endl;

        }

    };

    void display()

    {

        cout << "in display ns "

             << "a=" << a << endl; // namespace member

    }

}

int main()

{

    ns::display();

    ns::Sample s1;

    s1.display();

    ns::Sample s2(10);

    s2.display();

}

->nested namespace:

#include <iostream>

using namespace std;

namespace ns1

{

    int a = 500;

    float b = 50.5F;

    namespace ns2 // nested namespace

    {

        float add(int x, float y)

        {

            a = x;

            b = y;

            return (a + b);

        }

    }

}

int main()

{

    cout << "ns1::a=" << ns1::a << "ns1::b=" << ns1::b << endl;

    cout << ns1::ns2::add(10, 20.5) << endl; // invoke a function of a nested namespace

    cout << "ns1::a=" << ns1::a << "ns1::b=" << ns1::b << endl;

}

->using directive

#include <iostream>

using namespace std;

namespace ns1

{

    int a = 500;

    float b = 50.5F;

}

namespace ns2

{

    int a = 50;

    float b = 5.5F;

}

int main()

{

    using ns1::a;

    using ns1::b;

    cout << "ns1::a=" << a << "ns1::b=" << b << endl;

    // using ns2::a;    //ambiguity

    // using ns2::b;    //ambiguity

    cout << "ns2::a=" << ns2::a << "ns2::b=" << ns2::b << endl;

}

->priority of using declaration is greater than the using directive

#include <iostream>

using namespace std;

namespace ns1

{

    int a = 500;

    float b = 50.5F;

}

namespace ns2

{

    int a = 50;

    float b = 5.5F;

}

int main()

{

    using namespace ns1; // using directive

    cout << "ns1::a=" << a << "ns1::b=" << b << endl;

    using ns2::a; // priority of using declaration is greater than

    using ns2::b; // the priority of the using directive

    cout << "ns2::a=" << a << "ns2::b=" << b << endl;

}

->

#include <iostream>

using namespace std;

namespace ns1

{

    int a = 500;

    float b = 50.5F;

}

namespace ns2

{

    int a = 50;

    float b = 5.5F;

}

int main()

{

    using ns2::a; // priority of using declaration is greater than

    using ns2::b; // the priority of the using directive

    cout << "ns2::a=" << a << "ns2::b=" << b << endl;

    using namespace ns1; // error is not thrown, but ns1 values are not displayed unless fully qualified names are used

    cout << "ns1::a=" << a << "ns1::b=" << b << endl;   //didn't display the ns1's values

    cout << "ns1::a=" << ns1::a << "ns1::b=" << ns1::b << endl;

}

using ns2::fun; //we can import a part of the namespace ns2 in this way and then we can call the function.

->

#include <iostream>

using namespace std;

namespace ns1

{

    int a = 500;

    float b = 50.5F;

    void fun(int a = 0)

    {

        cout << "ns1 fun() " << a << endl;

    }

    namespace ns2

    {

        int a = 50;

        float b = 5.5F;

        void fun(int a)

        {

            cout << "ns2 fun() " << a << endl;

        }

    }

}

int main()

{

    using namespace ns1::ns2; // ok

    fun(10);                  // ok

    // using ns2::fun;  //ns2 has no independent existence

    // fun(5);

    using namespace ns1;

    fun();

    ns2::fun(100);

}

->namespace extension:

#include <iostream>

using namespace std;

namespace ns1

{

    int p, q;

}

namespace ns2

{

    int x, y;

}

namespace ns1 // namespace extension or discontiguous namespace

{

    int a, b;

}

int main()

{

    ns1::a = 1;

    ns1::b = 2;

    ns1::p = 10;

    ns1::q = 20;

    ns2::x = 100;

    ns2::y = 200;

    namespace one = ns1;    // namespace alias

    namespace two = ns2;    //namespace alias

    cout << ns1::a << " " << ns1::b << "  " << ns1::p << "  " << ns1::q << " "<<endl;

    cout << ns2::x << " " << ns2::y << "  "<<endl; // both the current name and new name will exist

    two::x = 500;

    two::y = 600;

    cout << two::x << " " << two::y << "  "<<endl;

}

->

#include<iostream>

#include<string.h>

using namespace std;

namespace ns1

{

    int p,q;

}

namespace ns2

{

    int p,q;

}

int main()

{

    using namespace ns1;

    p=10;

    q=20;

    using ns2::p;

    using ns2::q;

    p=4;         //using declaration will override the using directive imported entities

    q=5;

    cout<<"p="<<p<<"q="<<q<<endl;

    cout<<"p="<<p<<"q="<<q<<endl;

    cout<<"ns1::p="<<ns1::p<<"ns1::q="<<ns1::q;

}

-> unnamed namespace:

#include <iostream>

using namespace std;

namespace // unnamed namespace

{

    void fun(int x)

    {

        cout << "hi " << x;

    }

}

int main()

{

    fun(20);

}

->

#include <iostream>

using namespace std;

void fun(int x)

{

    cout << "global \n";

}

namespace // unnamed namespace

{

    void fun(int x = 0)

    {

        cout << "hi " << x;

    }

}

int main()

{

    ::fun(1); //global fun invoked

}