name with a using-declaration. When you use a using-directive, however, name resolution
takes place as if you declared the names in the smallest declarative region containing both
the using-declaration and the namespace itself. For the following example, that would be
the global namespace. If you use a using-directive to import a name that already is
declared in a function, the local name will hide the namespace name, just as it would hide
a global variable of the same name. However, you still can use the scope resolution
operator:

namespace Jill {
    double bucket(double n) { ... }  
    double fetch;
    struct Hill { ...};
}

char fetch; // global namespace
int main()
{
    using namespace Jill; // import all namespace names
    Hill Thrill; // create a type Jill::Hill structure
    double water = bucket(2); // use Jill::bucket();
    double fetch; // not an error; hides Jill::fetch
    cin >> fetch; // read a value into the local fetch
    cin >> ::fetch; // read a value into global fetch
    cin >> Jill::fetch; // read a value into Jill::fetch
    ...
}

int foom()
{
    Hill top; // ERROR
    Jill::Hill crest; // valid
}

Here, in main(), the name Jill::fetch is placed in the local namespace. It doesn't have local
scope, so it doesn't override the global fetch. But the locally declared fetch hides both
Jill::fetch and the global fetch. However, both of the last two fetch variables are available
if you use the scope resolution operator. You might want to compare this example to the
preceding one, which used a using-declaration.