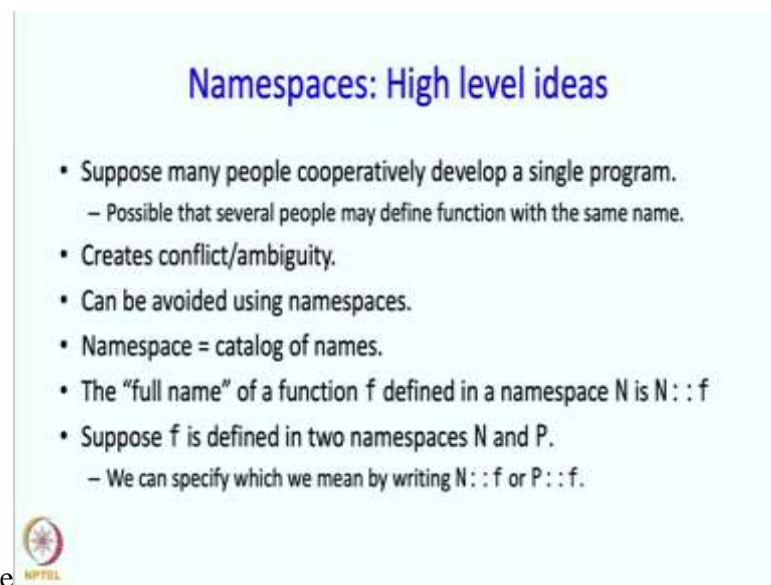


**An introduction to programming through C++**  
**Professor Abhiram G. Ranade**  
**Department of Computer Science and Engineering,**  
**Indian Institute of Technology, Bombay**  
**Lecture Number 13, Part-3**  
**Program Organization and Functions**  
**Namespaces**


Welcome back. In the last segment we discussed how to split a program over many files, functions declarations and definitions and header files. In this segment we are going to discuss Namespaces.

(Refer Slide Time: 00:37)



**Namespaces: High level ideas**

- Suppose many people cooperatively develop a single program.
  - Possible that several people may define function with the same name.
- Creates conflict/ambiguity.
- Can be avoided using namespaces.
- Namespace = catalog of names.
- The “full name” of a function  $f$  defined in a namespace  $N$  is  $N : : f$
- Suppose  $f$  is defined in two namespaces  $N$  and  $P$ .
  - We can specify which we mean by writing  $N : : f$  or  $P : : f$ .

e 

So, suppose many people cooperatively develop a single program. Now, it is possible that several people may define functions with the same name. So, it is a little bit cumbersome for you to coordinate if you are writing a small function that you say, oh I am going to use a function called  $F$ , you please do not use a function called  $F$  as well. So, it is so I that cooperation is that coordination is cumbersome and if you do not have the coordination then you may have some name conflicts. So, how do we avoid this? This can be avoided using something called as Namespace. A Namespace is basically a catalog of names. So, effectively the “full name” of a function  $F$  defined in the namespace  $N$  is  $N::F$ . So, if  $F$  is defined in two namespaces  $N$  and  $P$  then,  $N$  we can specify which one we mean by writing  $N::F$  or  $P::F$  and there are other ways of specifying as well which we will see soon.

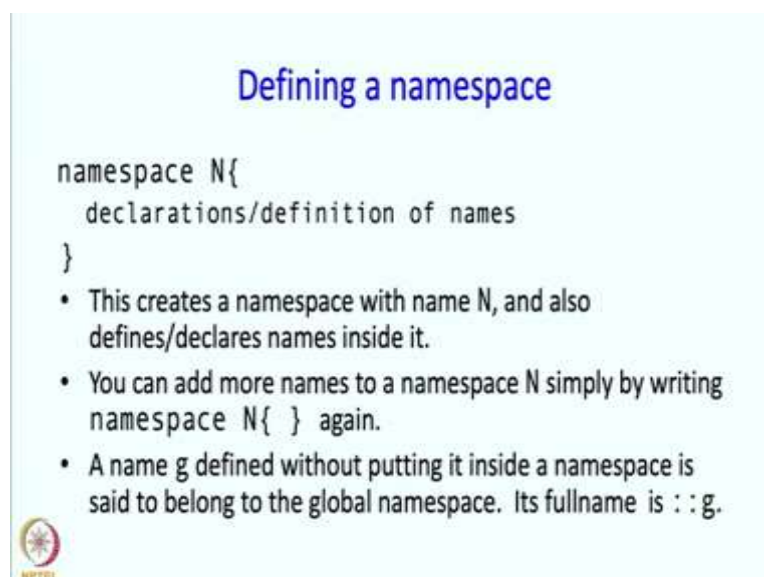
But basically now, we can have unambiguous ways of referring to different functions even if their name is same, because their full name is going to be different. So, effectively each programmer could use a different name space if there is need.

(Refer Slide Time: 02:21)



So, how do you define a Namespace? Declaration is simple, defining a namespace is simple, you just write namespace followed by the name of that namespace. In this case the name is N and then inside that block you put in declarations or definitions of names.

(Refer Slide Time: 02:41)



This creates a namespace with the name N and also defines or declares names inside it. You can add more names into a name space, simply by writing namespace N again and putting


names inside that new block in the new position. A name `g` which is defined without putting it inside a namespace is said to belong to the global namespace. So, all the names that you have been defining so far are in the global namespace. So, where full name is `::g`. So, you could refer to then in this way but if you wish.

(Refer Slide Time: 03:27)

### Defining a namespace

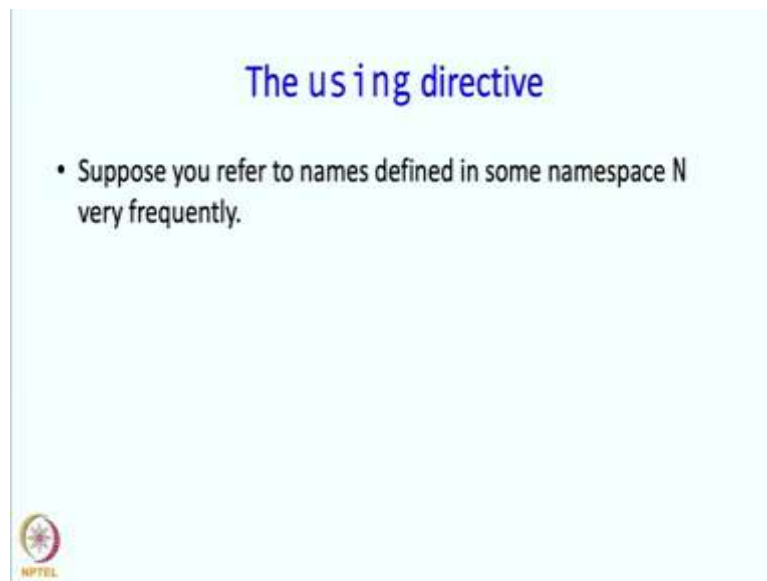
```
namespace N{
  declarations/definition of names
}
```

- This creates a namespace with name `N`, and also defines/declares names inside it.
- You can add more names to a namespace `N` simply by writing `namespace N{ }` again.
- A name `g` defined without putting it inside a namespace is said to belong to the global namespace. Its fullname is `::g`.



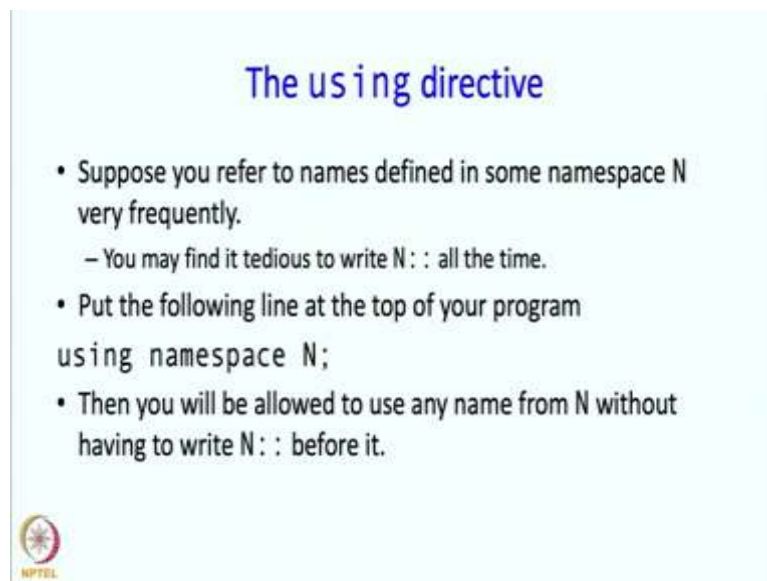
So, here is an example of using the namespace. So, first I'm defining a namespace `N` and it contains declaration actually, it contains definition of `g` series. So I have put (...) to say that entire definition, the entire body will go there. Similarly, there is the definition of `lcm`. Now, in my main program, I need to write `N::lcm` because I want to access this `lcm` which is defined in this namespace `N`.

(Refer Slide Time 04:10)



Now, there is something called the ‘using’ directive which comes in handy often. So, if you have defined a namespace N and you have many names defined in it and you want to access them quite frequently, so you may get tired of writing N:: all the time.

(Refer Slide Time: 04:31)




So, what you might do is you put the following (line), you might put the following line at the top of your file. So, I say using namespace N, this is the line you put. Subsequently or for all other references in that file, you will be allowed to use any name from N without having to write N:: before it. So, N:: will become implicit.

(Refer Slide Time: 04:58)

```
Example using using

namespace N{
    int gcd(int m, int n){ ... }
    int lcm(int m, int n){ ... }
}
using namespace N;
int main(){
    cout << lcm(36,24) << endl;
}

```




So, if I write this directive then the previous program becomes like this. I will write namespace N int gcd int lcm and then I have this using namespace over here. So, this using namespace says when I use lcm over here it is implicitly interpreted as N::lcm and I will get to the definition which is given in this namespace.

(Refer Slide Time: 05:31)

```
What we discussed
```

- Namespaces
  - Helps many people use the same name and yet link their work together if needed
- The using directive
- Next:
  - How to use C++ without simplecpp
  - Concluding remarks on this lecture sequence



Alright, so what have we discussed, we have said that namespaces help many people to use the same name and yet combine their work together if needed, then we discussed the using directive, okay. In the next segment we are going to use C++ without simple CPP and we are also going to conclude this entire lecture sequence. So, let us take a break.