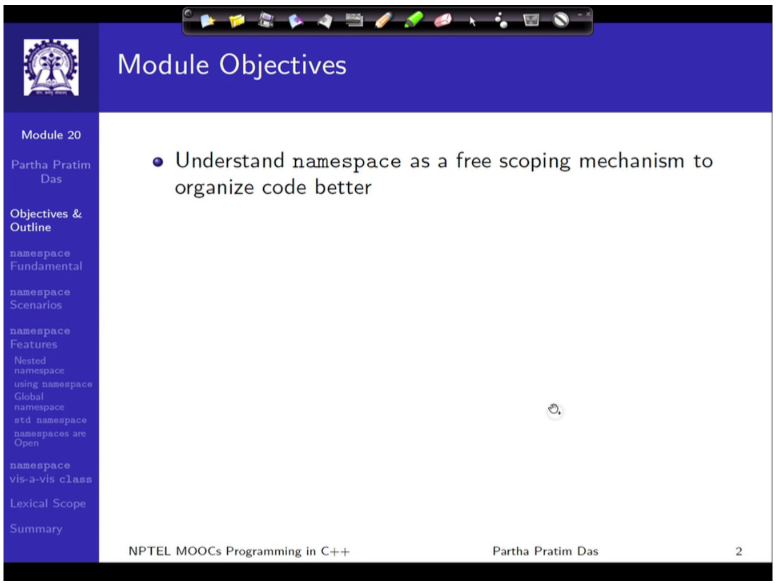
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 35
Namespace

Welcome to Module 20 of Programming in C++. In this module we will talk about Namespaces.

(Refer Slide Time: 00:28)



Module Objectives

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features
- Nested namespace
- using namespace
- Global namespace
- std namespace
- namespaces are Open

namespace vis-a-vis class

Lexical Scope

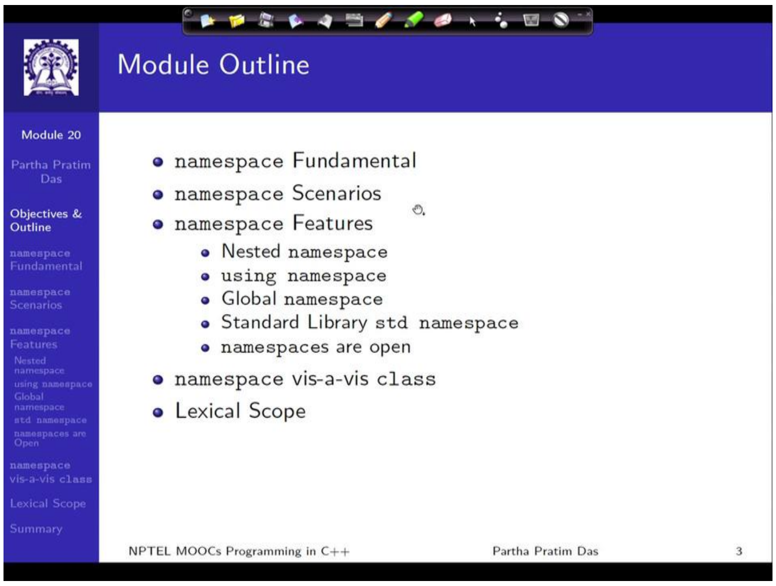
Summary

- Understand namespace as a free scoping mechanism to organize code better

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

Namespaces are a concept of lexical scoping, of which we have variety of options that you already know in C++. But, we will expose you to this additional concept of scoping and how does it help in organising code structure.

(Refer Slide Time: 00:51)



Module Outline

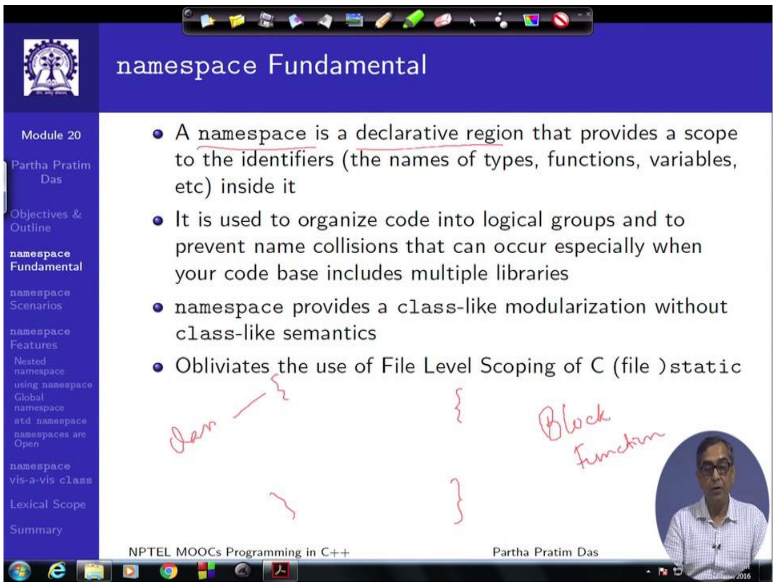
- namespace Fundamental
- namespace Scenarios
- namespace Features
  - Nested namespace
  - using namespace
  - Global namespace
  - Standard Library std namespace
  - namespaces are open
- namespace vis-a-vis class
- Lexical Scope

Module 20 | Partha Pratim Das | Objectives & Outline | namespace Fundamental | namespace Scenarios | namespace Features | Nested namespace | using namespace | Global namespace | std namespace | namespaces are Open | namespace vis-a-vis class | Lexical Scope | Summary

NPTEL MOOCs Programming in C++ | Partha Pratim Das | 3

This is the outline, and you will find it on the left of the every slide as we go forward.

(Refer Slide Time: 01:00)



namespace Fundamental

- A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it
- It is used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries
- namespace provides a class-like modularization without class-like semantics
- Obliviates the use of File Level Scoping of C (file)static

Module 20 | Partha Pratim Das | Objectives & Outline | namespace Fundamental | namespace Scenarios | namespace Features | Nested namespace | using namespace | Global namespace | std namespace | namespaces are Open | namespace vis-a-vis class | Lexical Scope | Summary

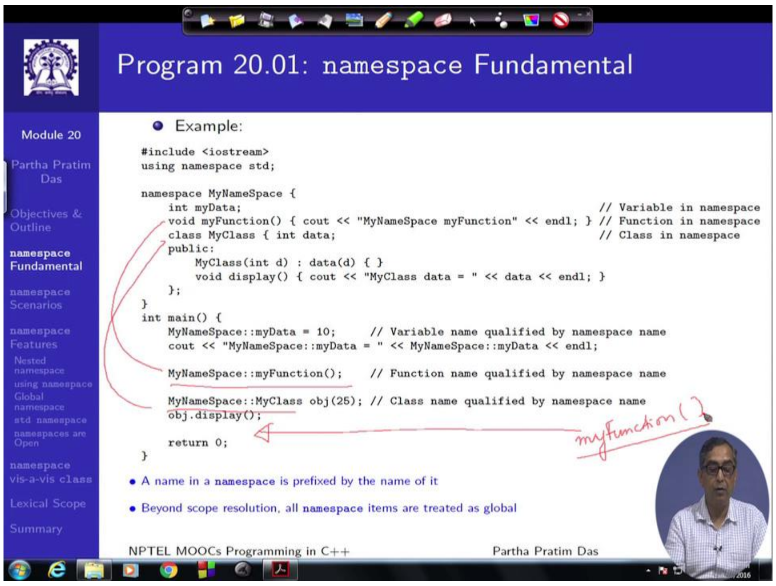
NPTEL MOOCs Programming in C++ | Partha Pratim Das

So, first let me introduce what is a namespace. A namespace, as I said is a declarative region; it is a scope, like this. So, we already are aware of scope like block, we know that every function has a scope; we know class has a scope, class some, class name has a scope. So, namespace is something absolutely similar to that. Here we have a scope it is a declarative region, within which I can have variety of different identifiers, variety of different symbols. We have types, functions, variables, classes, and other namespaces and so on.

The major purpose as I said is to organise the coding into logical groups. And that is a that is a very critical requirement. And, one of the major reasons that namespace exists is to prevent name clash, name collision that can occur. Especially, when the code base include multiple libraries, the code base intends to use names which are reserved by the third party libraries or standard library, or code base evolves has been developed by independent developers and inadvertently the same set of names have got used. So, this is the main purpose off, this is what namespace is, it defines a scope and this is the main purpose to organise the code. Namespace provides a class like modularization, like we know that every class definition is kind of a modular definition, but the difference is it does not have any semantics. It is just a scoping rule, class also does scoping, but it does scoping with class semantics. Namespace separately will not have any semantics.

And, for those of you who are familiar with file scope in C, like which we say is a file static variables or file static functions, use of namespace will obliviate the requirement of file scope, as is used in C. So, if you have been using the same, then whenever you need that, you should actually not use this and rather use namespace in that place.

(Refer Slide Time: 03:33)



Program 20.01: namespace Fundamental

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- Example:

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

namespace MyNameSpace {
    int myData;                                                  // Variable in namespace
    void myFunction() { cout << "MyNameSpace myFunction" << endl; } // Function in namespace
    class MyClass { int data;                                    // Class in namespace
    public:
        MyClass(int d) : data(d) { }
        void display() { cout << "MyClass data = " << data << endl; }
    };
}
int main() {
    MyNameSpace::myData = 10;      // Variable name qualified by namespace name
    cout << "MyNameSpace::myData = " << MyNameSpace::myData << endl;

    MyNameSpace::myFunction();     // Function name qualified by namespace name

    MyNameSpace::MyClass obj(25); // Class name qualified by namespace name
    obj.display();

    return 0;
}
```

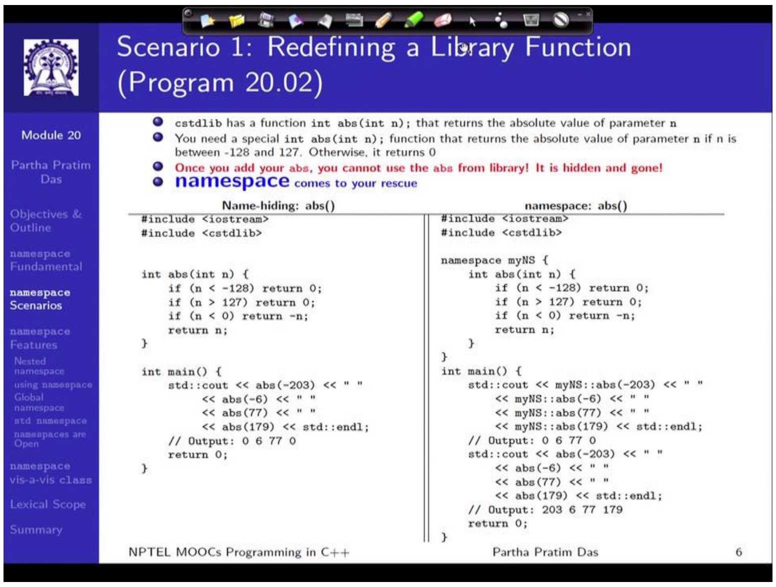
- A name in a namespace is prefixed by the name of it
- Beyond scope resolution, all namespace items are treated as global

NPTEL MOOCs Programming in C++ Partha Pratim Das

Let me introduce you to the actual code as to how to write this, this is not otherwise a meaningful example. So here, I am saying namespace is a keyword with which like the class keyword you do and that should be followed by the namespace name. So, the syntax is pretty much like how you define a class. It has an associated scope in terms of matching curly braces, and anything that you write within that namespace is belongs to the namespace, which means that any symbol that you write inside this gets qualified by the name of the namespace.

So, here I show 3 different kinds of entries. One is a name of a variable, my data, one is a function and one is a class, my data, my function, my class. And then, in the main I show how to use that. If I want to use the data, then I need to write it in this form. So, you can see that the variable name was my data that has got qualified by the namespace name. This is exactly how class names qualify, for example, I would immediately remind you about static data members in classes this is how you write the static data members in classes. So, the namespace name followed by the symbol name, separated by the scope resolution operator is a way to refer to the namespace object. For example, for the function that we have above, this name is, for the class, this name. In this context of the namespace, here if I simply write my function and try to invoke that then I will get compilation error. Because, there is nothing called no symbol called my function in this program. The, my function symbol is within the namespace and therefore will always have to be prefixed with the namespace. So, this is a basic way you define namespaces and you use namespaces.

(Refer Slide Time: 05:43)



Scenario 1: Redefining a Library Function (Program 20.02)

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- cstdlib has a function `int abs(int n);` that returns the absolute value of parameter `n`
- You need a special `int abs(int n);` function that returns the absolute value of parameter `n` if n is between -128 and 127. Otherwise, it returns 0
- Once you add your abs, you cannot use the abs from library! It is hidden and gone!
- **namespace** comes to your rescue

Name-hiding: abs()

```
#include <iostream>
#include <cstdlib>


int abs(int n) {
    if (n < -128) return 0;
    if (n > 127) return 0;
    if (n < 0) return -n;
    return n;
}

int main() {
    std::cout << abs(-203) << " "
         << abs(-6) << " "
         << abs(77) << " "
         << abs(179) << std::endl;
    // Output: 0 6 77 0
    return 0;
}
```

namespace: abs()

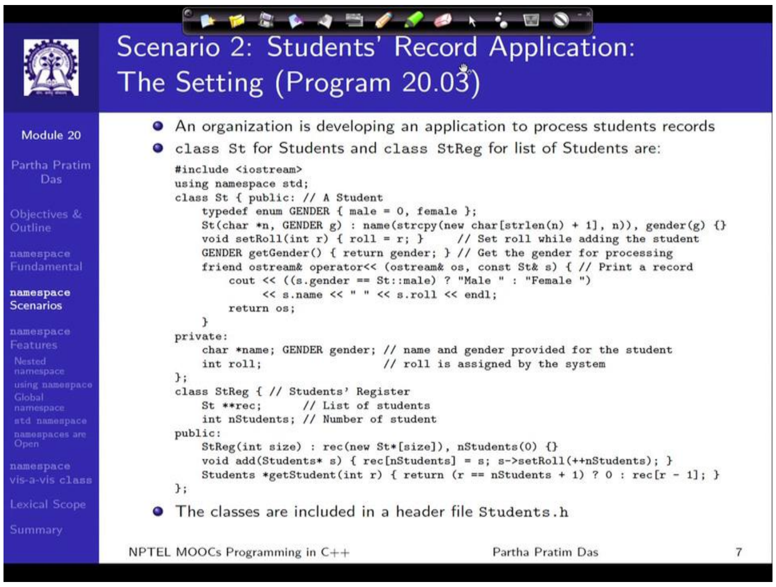
```
#include <iostream>
#include <cstdlib>

namespace myNS {
    int abs(int n) {
        if (n < -128) return 0;
        if (n > 127) return 0;
        if (n < 0) return -n;
        return n;
    }
}
int main() {
    std::cout << myNS::abs(-203) << " "
        << myNS::abs(-6) << " "
        << myNS::abs(77) << " "
        << myNS::abs(179) << std::endl;
    // Output: 0 6 77 0
    std::cout << abs(-203) << " "
        << abs(-6) << " "
        << abs(77) << " "
        << abs(179) << std::endl;
    // Output: 203 6 77 179
    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

Let us look at two scenarios, one simple and one little bit more involved. Here is the scenario where I am trying to redefine the library function. So, all of us know that library function has an ABS function, standard library has a ABS function which finds the absolute value, but I want to give a different behaviour to that. I want to say that within minus 128 and 127 it will find the absolute, but otherwise if it is outside that range then it will return a zero. So, the simple way to do this, I define the abs function and start using it, and the flip side of this is if we do that then, the abs function that exists in the C standard library is hidden, that is once I have defined my abs function, then the abs function that is available from the library is no more available. So if I just use abs, it will always mean my abs it will never mean the abs that existed in the library. So, I lose by this if I do this in C and of course, then we can do it in C++ I lose the ability to actually refer to the original library function.

So, namespace provides a nice solution to that. If I want to do this and also want to keep referring to the library function, I can still define my abs function, but I will just put it in a new namespace. So, I have given the name of that as my n s. So, with that if I refer to the abs function as my n s colon colon abs, then it refers to this function. But if I just say abs, since there is no abs available because this particular new version of abs is within my n s name scope. So, if I just say abs then it means the abs of the one that exists in the standard library. In this way I can protect my new introduced definitions without clashing with the library name that exists. This is one very typical requirement that you will often face and this is a solution using namespace is a good solution.

(Refer Slide Time: 08:04)



Scenario 2: Students' Record Application: The Setting (Program 20.03)

- An organization is developing an application to process students records
- class St for Students and class StReg for list of Students are:

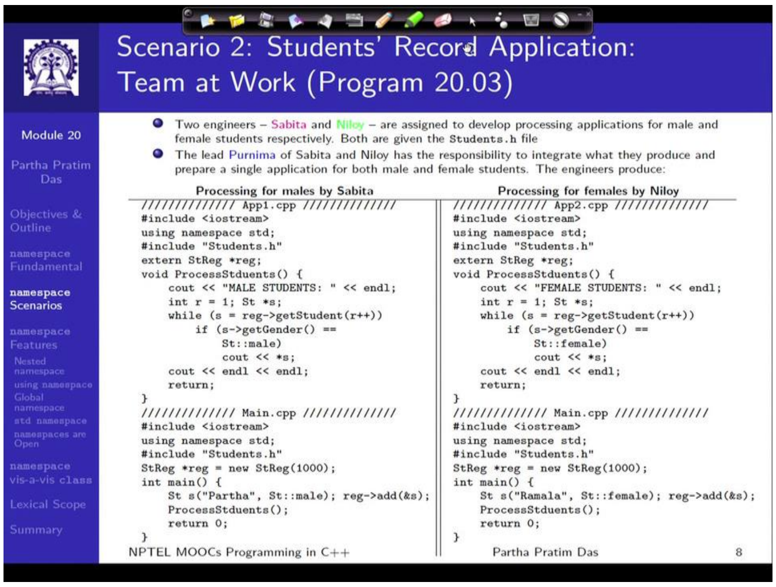
```
#include <iostream>
using namespace std;
class St { public: // A Student
    typedef enum GENDER { male = 0, female };
    St(char *n, GENDER g) : name(strcpy(new char[strlen(n) + 1], n)), gender(g) {}
    void setRoll(int r) { roll = r; }     // Set roll while adding the student
    GENDER getGender() { return gender; } // Get the gender for processing
    friend ostream& operator<< (ostream& os, const St& s) { // Print a record
        cout << ((s.gender == St::male) ? "Male " : "Female ")
             << s.name << " " << s.roll << endl;
        return os;
    }
private:
    char *name; GENDER gender; // name and gender provided for the student
    int roll;                  // roll is assigned by the system
};
class StReg { // Students' Register
    St **rec;      // List of students
    int nStudents; // Number of student
public:
    StReg(int size) : rec(new St*[size]), nStudents(0) {}
    void add(Students* s) { rec[nStudents] = s; s->setRoll(++nStudents); }
    Students *getStudent(int r) { return (r == nStudents + 1) ? 0 : rec[r - 1]; }
};
```

- The classes are included in a header file Students.h

NPTEL MOOCs Programming in C++ Partha Pratim Das 7

Let me quickly run through a development scenario. This is what very regularly happens in any organisation. Suppose, an organisation is developing an application to process students' records and let us say there are two classes; one for the students. So, we have two classes; this represents a student and this represents a list of students. You may not really bother about the details though it actually is a correct program. So, you could at a later point of time read through and actually find what this program is doing. It certainly has different, it has the constructor, it has a sums and get functions and it has a output operator to be able to write a student record. And, this has a list and it keeps track of the number of students that exist and it has an option to add a student to this list. So, whenever you add a student the roll number of that student gets allocated. In terms of developing this processing application, what the organisation does and that is very typical that somebody senior of the designers will possibly have designed this classes. And then the task is given to multiple developers to develop different parts of the system. So, these classes are designed and given in a header file, student’s dot h.

(Refer Slide Time: 09:28)



Scenario 2: Students' Record Application: Team at Work (Program 20.03)

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- Two engineers – Sabita and Niloy – are assigned to develop processing applications for male and female students respectively. Both are given the Students.h file
- The lead Purnima of Sabita and Niloy has the responsibility to integrate what they produce and prepare a single application for both male and female students. The engineers produce:

Processing for males by Sabita

```
////////////// App1.cpp //////////////
#include <iostream>
using namespace std;
#include "Students.h"
extern StReg *reg;
void ProcessStduents() {
    cout << "MALE STUDENTS: " << endl;
    int r = 1; St *s;
    while (s = reg->getStudent(r++))
        if (s->getGender() ==
            St::male)
            cout << *s;
    cout << endl << endl;
    return;
}
////////////// Main.cpp //////////////
#include <iostream>
using namespace std;
#include "Students.h"
StReg *reg = new StReg(1000);
int main() {
    St s("Partha", St::male); reg->add(&s);
    ProcessStduents();
    return 0;
}
```

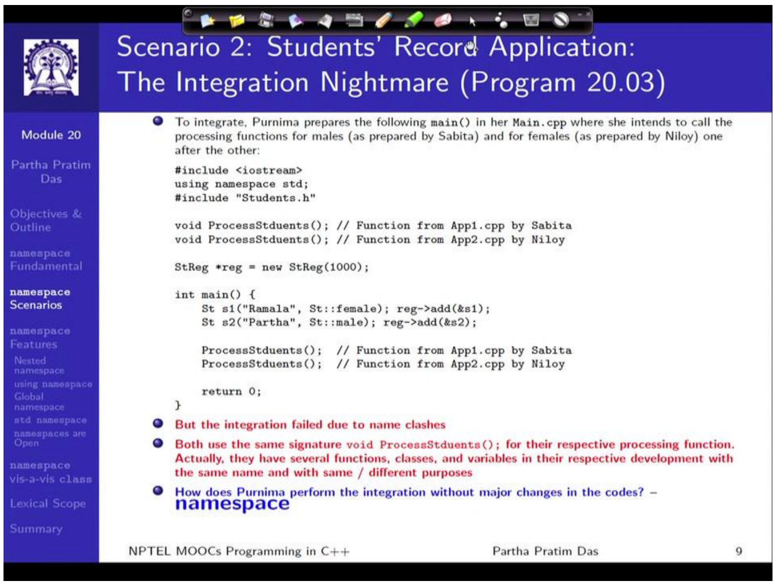
Processing for females by Niloy

```
////////////// App2.cpp //////////////
#include <iostream>
using namespace std;
#include "Students.h"
extern StReg *reg;
void ProcessStduents() {
    cout << "FEMALE STUDENTS: " << endl;
    int r = 1; St *s;
    while (s = reg->getStudent(r++))
        if (s->getGender() ==
            St::female)
            cout << *s;
    cout << endl << endl;
    return;
}
////////////// Main.cpp //////////////
#include <iostream>
using namespace std;
#include "Students.h"
StReg *reg = new StReg(1000);
int main() {
    St s("Ramala", St::female); reg->add(&s);
    ProcessStduents();
    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

At this point the responsibility is given to; let us say to engineers, by dividing the development, separately for male students and for female students. They will have different requirements of hostel, different requirements of subject classification and so on. So, this is what is being done by Savitha, for male students; similar development being done by Niloy, for female students; and Poornima is their lead, who finally has to integrate both these applications into your final applications. So what happens is, it may be incidental, coincidental, whatever, is Savitha calls this whole processing application that she is doing for the male students as process student; and, Niloy also chooses the same name of the function. And then, they write independent main applications to test out whatever they have developed. You need not bother about what this is doing, this basically is, if this application prints the male students, this application prints the female students, but that is just an indicative one I am not really concerned about what the processing is happening. But the fact is that, they have chosen incidentally the same function name and independently developed it together. Now, both of them submit the code to Poornima, the lead, now Poornima has to put them together.

(Refer Slide Time: 10:56)



Scenario 2: Students' Record Application: The Integration Nightmare (Program 20.03)

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- To integrate, Purnima prepares the following `main()` in her `Main.cpp` where she intends to call the processing functions for males (as prepared by Sabita) and for females (as prepared by Niloy) one after the other:

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

void ProcessStduents(); // Function from App1.cpp by Sabita
void ProcessStduents(); // Function from App2.cpp by Niloy

StReg *reg = new StReg(1000);

int main() {
    St s1("Ramala", St::female); reg->add(&s1);
    St s2("Partha", St::male); reg->add(&s2);

    ProcessStduents();  // Function from App1.cpp by Sabita
    ProcessStduents();  // Function from App2.cpp by Niloy

    return 0;
}
```

- But the integration failed due to name clashes
- Both use the same signature `void ProcessStduents();` for their respective processing function. Actually, they have several functions, classes, and variables in their respective development with the same name and with same / different purposes
- How does Purnima perform the integration without major changes in the codes? – **namespace**

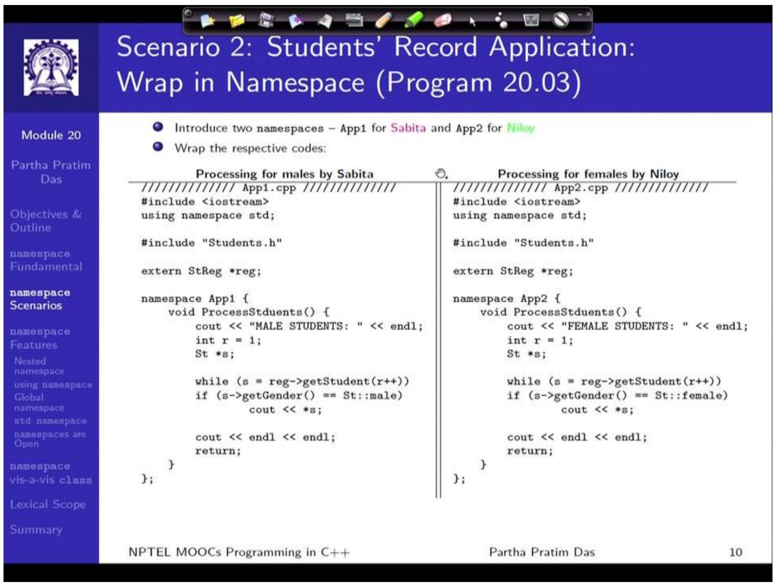
NPTEL MOOCs Programming in C++ Partha Pratim Das 9

So, certainly, if we just refer back once, Poornima will neither use this main, nor will use this main, because she has to integrate both of this codes. So, she will have to write a main unified application of for all. So, she tries to write that. So, she has to call these functions, the process students' functions. She has to put their definitions on top and she finds that, both of them use the same function name. So, if she would copy their codes, then this is what it will look like this is the same. As you can understand that this will give a name clash and it cannot be same, it will simply not compile. Now, whole integration process has failed.

I have shown in terms of a function name just as a indicative one, but in reality, certainly, the application would have would be in couple of thousands of lines. There could be several symbols, several functions, global variables, class names, several types and so on, which may have the same name between the two developers. There could be more developers also. There could be that in some cases, the names that are same between two developers' programs or code means the same thing; in some cases, same name as used by Savitha, will be used by Niloy in a different meaning. So, this is a very very difficult problem. If Poornima has to integrate, she has to understand the whole code and then edit that make changes within that and that will become a complete nightmare. So, this is what is known as the integration nightmare that typically happens in an organisation.

But, the other side of the fact is independently both the applications as developed by Savitha for the male students and by Niloy for the female students independently they work, it’s only that the, they are sharing certain common names and therefore namespace turns out to be a panacea which can solve this problem very easily.

(Refer Slide Time: 13:15)



Scenario 2: Students' Record Application: Wrap in Namespace (Program 20.03)

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- Introduce two namespaces – App1 for Sabita and App2 for Niloy
- Wrap the respective codes:

Processing for males by Sabita

```
////////////// App1.cpp //////////////
#include <iostream>
using namespace std;

#include "Students.h"

extern StReg *reg;

namespace App1 {
    void ProcessStduents() {
        cout << "MALE STUDENTS: " << endl;
        int r = 1;
        St *s;

        while (s = reg->getStudent(r++))
        if (s->getGender() == St::male)
                cout << *s;

        cout << endl << endl;
        return;
    }
};
```

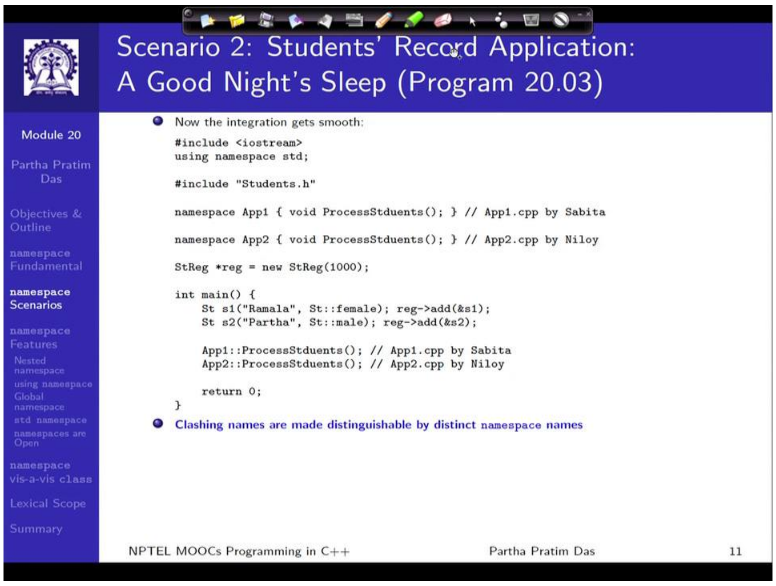
Processing for females by Niloy

```
////////////// App2.cpp //////////////
#include <iostream>
using namespace std;

#include "Students.h"

extern StReg *reg;

namespace App2 {
    void ProcessStduents() {
        cout << "FEMALE STUDENTS: " << endl;
        int r = 1;
        St *s;

        while (s = reg->getStudent(r++))
        if (s->getGender() == St::female)
                cout << *s;

        cout << endl << endl;
        return;
    }
};
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

All that Poornima now has to do is to take the applications that Savitha had developed and Niloy had developed and put them into two different namespaces. She decides on two names, app 1 and app 2 and puts Savitha’s developments in this, puts Niloy’s developments in this. So, once that has been done, then within this application, it is all within the same namespace. So, the application still works. This part will still work, but when you are outside when you are looking at from main functions point of view these two are into two different namespaces. So, they are basically, different functions.

(Refer Slide Time: 14:01)



Scenario 2: Students' Record Application: A Good Night's Sleep (Program 20.03)

- Now the integration gets smooth:

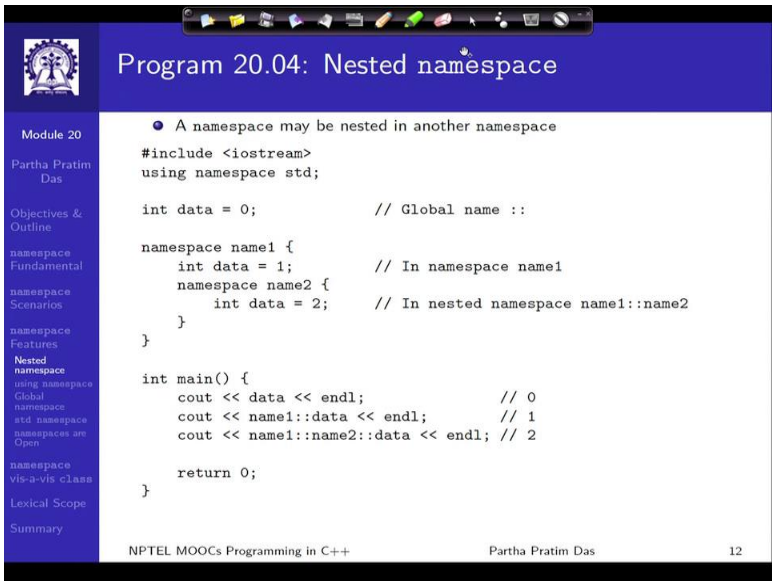
```
#include <iostream>
using namespace std;

#include "Students.h"

namespace App1 { void ProcessStduents(); } // App1.cpp by Sabita

namespace App2 { void ProcessStduents(); } // App2.cpp by Niloy

StReg *reg = new StReg(1000);

int main() {
    St s1("Ramala", St::female); reg->add(&s1);
    St s2("Partha", St::male); reg->add(&s2);

    App1::ProcessStduents(); // App1.cpp by Sabita
    App2::ProcessStduents(); // App2.cpp by Niloy

    return 0;
}
```

- Clashing names are made distinguishable by distinct namespace names

So, then she will come back after putting encapsulating them into namespaces. Now, she is back into the integration. This is, Savitha's applications is now quite processing students function in app 1 namespace; for Niloy, that is in the app 2 namespace. So, all that she needs to do is, by doing this, she has resolved the clash of name between the two development units and in terms of the application, all that she needs to do is to use the namespace prefix and call the two functions independently, one after the other or whatever she wants to do.

So, this is a very typical approach, which can solve a huge amount of practical integration problem and that needs to be kept in mind. Of course, it is not a good idea that multiple developers in the same system will not be able to coordinate and resolve to have distinct names, but it often is a good idea to use namespace also to make different modules so that in between different modules you will not have to really bother about what names, what different supporting functions, supporting classes and all that, you may be using. So, if we are doing a student record development, there could be one module which deals primarily with the students' academic part; one module deals with the students fees; one module deals with the students hostel; another module will deal with their timetable and so on. So, a good way to organise the code and to the design would be to assign different namespaces to this modules and basically separate out the code in that manner, so that you make sure that it will never clash among themselves.

(Refer Slide Time: 15:43)



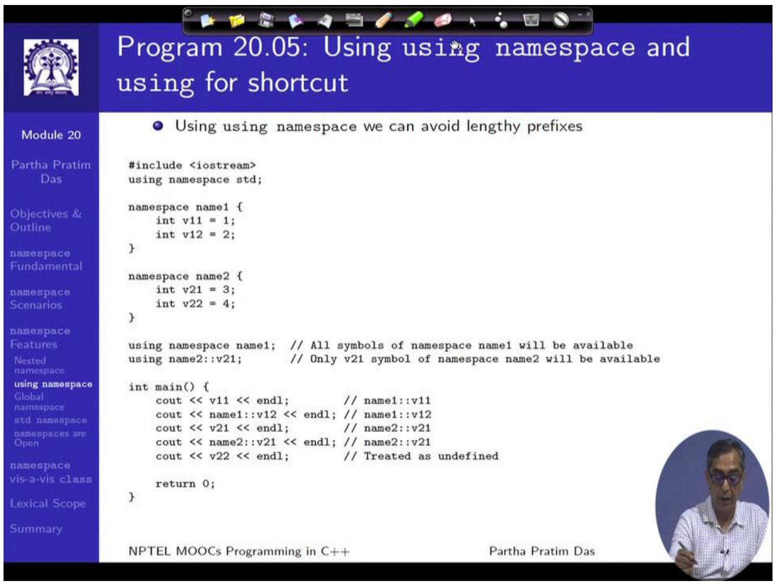
Program 20.04: Nested namespace

- A namespace may be nested in another namespace

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

int data = 0;               // Global name ::

namespace name1 {
    int data = 1;           // In namespace name1
    namespace name2 {
        int data = 2;       // In nested namespace name1::name2
    }
}

int main() {
    cout << data << endl;                 // 0
    cout << name1::data << endl;          // 1
    cout << name1::name2::data << endl; // 2

    return 0;
}
```

So, let me just to complete the discussion you have understood why namespace is important. So, let me just go over the quick features that a namespace has. As I said, it is very similar to class; like class, the namespaces can be nested. So, I have a namespace name 1 here and within that I have another namespace. So, all that happens is, if I nest one namespace into another, the name of this nested namespace is qualified by the outer namespace. So, that is the simple thing. So, which means that this data is in the namespace name 1 whereas, this data is in the namespace of name 1 colon colon name 2, because this itself is name 1 colon colon name 2. So, if we write this code, if I just write data then, data is only outside so it will mean this data. If I write name 1 colon colon data it means this data, if I write name 1 colon colon name 2 colon colon data, it means this data. So, as you keep on nesting namespaces, more and more prefix get added. So, any level of nesting is possible for namespaces.

(Refer Slide Time: 16:56)



Program 20.05: Using using namespace and using for shortcut

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- Using using namespace we can avoid lengthy prefixes

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

namespace name1 {
    int v11 = 1;
    int v12 = 2;
}

namespace name2 {
    int v21 = 3;
    int v22 = 4;
}

using namespace name1;  // All symbols of namespace name1 will be available
using name2::v21;       // Only v21 symbol of namespace name2 will be available

int main() {
    cout << v11 << endl;        // name1::v11
    cout << name1::v12 << endl; // name1::v12
    cout << v21 << endl;        // name2::v21
    cout << name2::v21 << endl; // name2::v21
    cout << v22 << endl;        // Treated as undefined

    return 0;
}
```

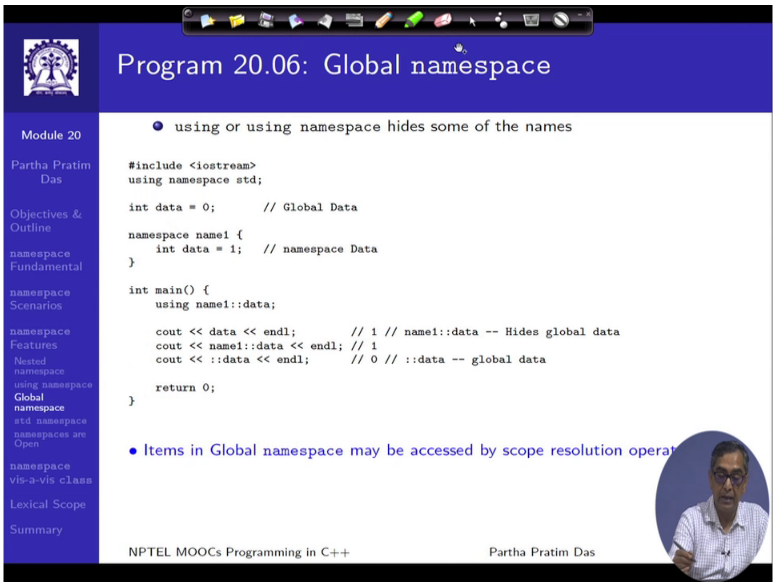
NPTEL MOOCs Programming in C++ Partha Pratim Das

Often, if you do it like this then it might become difficult that as you have multiple nestings and multiple symbols that you want to use, every time you will have to put the namespace name. So, there is a shortcut that you can use. There are basically two shortcuts that you can use; one is these are called using. So, one shortcut that you can do is, you say using and then you put a name of a namespace. So, you say using namespace name 1. What this will mean is any symbol that is subsequently used after this; you will try to check if this namespace name 1 has that symbol. And if it has that symbol then you will be referring to that symbol. So, this is the feature of using namespace.

The other using feature is, you can say using and you can say that, actually a qualified symbol itself. So, if you specify a qualified symbol, name qualified symbol then whenever later on you just talk about the symbol name, it will mean this qualified symbol. So, this example should help. So, I have two namespaces, name 1 and name 2; name 1 has two symbols; name 2 as two symbols; and, I have a using on name 1, the namespace name 1 and I have a using on this particular symbol, variable of name 2. So, what happens? If I say v 1 1, what it will check? It will check that I am using namespace name 1. So, does v 1 1 exists in that namespace; it does. So, it associates with this. If I say, name 1 colon colon v 1 2, it will associate here. So, you can see that even when I am doing using it is not mandatory that I will have to use the short form. In place of saying this, I could have simply written v 1 2; that also would have referred to the same variable in name 1, because I have a using namespace name 1. But, it is not mandatory I can use the using shortcut or I can also use the fully qualified name, as I have done here. Think about v 2 1. V 2 1 will mean this. Why? I do not have a using on namespace name 2. But, I have been using on this particular symbol itself. So, if I say v 2 1 it means that, it is name 2 colon colon v 2 1 and it will be right.

Similarly, I can also directly still refer to this, by saying it is name 2 colon colon v 2 1; this also is permitted; explicit use of name is permitted. Think of v 2 2; v 2 2 belongs here. I do not have a using on name 2 namespace. So, v 2 2 cannot mean this one, this particular entity neither I have a using on, like name 2 colon colon v 2 2; I do not have that. So, the compiler cannot see any v 2 2 symbol in this code and the compiler will treat this as undefined. It will say, there is no symbol called v 2 2. This a basic use of using; as such, you have been seeing this in the whole code so far I had mentioned at the very beginning that, we will keep on writing this because all of the standard library symbols are in the namespace Std. So, writing this makes our life easier otherwise, your c out will have to be written as std colon colon c out; c in as std colon colon c in and so on.

(Refer Slide Time: 21:06)



Program 20.06: Global namespace

- using or using namespace hides some of the names

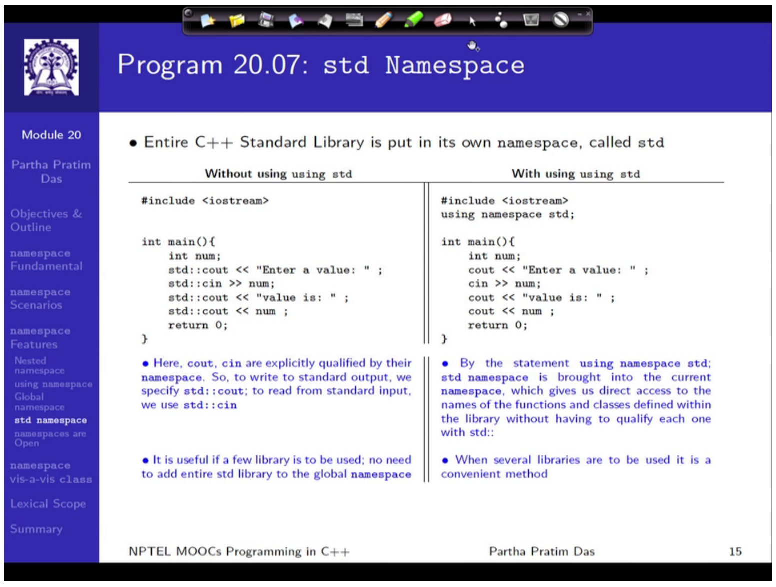
```
#include <iostream>
using namespace std;

int data = 0;       // Global Data

namespace name1 {
    int data = 1;   // namespace Data
}

int main() {
    using name1::data;

    cout << data << endl;         // 1 // name1::data -- Hides global data
    cout << name1::data << endl; // 1
    cout << ::data << endl;       // 0 // ::data -- global data

    return 0;
}
```

- Items in Global namespace may be accessed by scope resolution operat

Now, let us also talk about the global namespace. Suppose, I have an example. Let us understand the example; I have a variable written in the global scope data. In the namespace name 1, I have a variable data. Now, as such they are resolvable if you do not think about this program given below, they are resolvable, because if I write data, it means this and, if I write name 1 colon colon data then it means this, clear. But, let us suppose in this function main, I have a using for name 1 colon colon data, right. So, what does it mean? It means that, now if I talk about data, it means this data; it does not mean this data anymore; because, I have a using on name 1 colon colon data. So, it says that, name 1 colon colon data will be known as a data in, from this point onwards. So, if I say data, I get this.

If I say name 1 colon colon data, I also get this. So, what it means is that, I have lost the ability to get access to the data which was not defined in the namespace, which is defined outside. So, C++ gives us a mechanism to be able to access those symbols, which are in the global space, which are in the global scope. All that you do, you use the same notation, but just that the global, the consideration is kind of that, the global scope also as if at a namespace, but that namespace has no name. So, it is just a blank name. So, all that you do is, just put colon colon data. So, that will always mean the name in the global scope. So, that is a basic concept of the global namespace that exists.

(Refer Slide Time: 23:15)



Program 20.07: std Namespace

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features
- Nested namespace
- using namespace
- Global namespace
- **std namespace**
- namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- Entire C++ Standard Library is put in its own namespace, called std

Without using using std

```
#include <iostream>


int main(){
    int num;
    std::cout << "Enter a value: " ;
    std::cin >> num;
    std::cout << "value is: " ;
    std::cout << num ;
    return 0;
}
```

- Here, cout, cin are explicitly qualified by their namespace. So, to write to standard output, we specify std::cout; to read from standard input, we use std::cin

- It is useful if a few library is to be used; no need to add entire std library to the global namespace

With using using std

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

int main(){
    int num;
    cout << "Enter a value: " ;
    cin >> num;
    cout << "value is: " ;
    cout << num ;
    return 0;
}
```

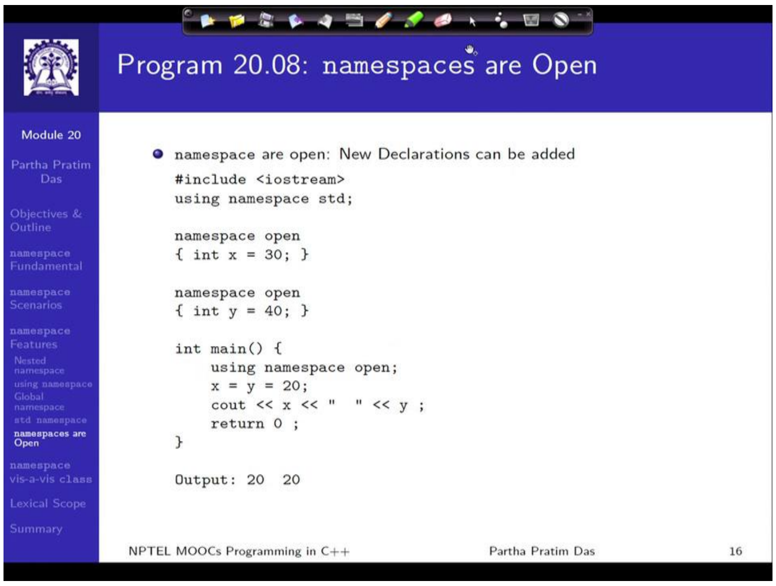
- By the statement using namespace std; std namespace is brought into the current namespace, which gives us direct access to the names of the functions and classes defined within the library without having to qualify each one with std::

- When several libraries are to be used it is a convenient method

NPTEL MOOCs Programming in C++ Partha Pratim Das 15

Standard namespace, so we have been talking about this that, all C++ puts all its standard library symbols, classes, functions, everything in the std namespace. So, if we just include IO stream and want to do, write a program using this, then every time we will have to prefix the symbol with the std colon colon. For example, if I have to write endl, I have to write it as std colon colon endl, endline. So, we can do a shortcut by putting this using namespace std. What will mean that, if I write C out, it will also check if std namespace has a C out; it does have one. So, std will relate to that. So, std namespace is the most important namespace that is available to us. There are couple of more namespaces also defined. We will talk about those later.

(Refer Slide Time: 24:09)



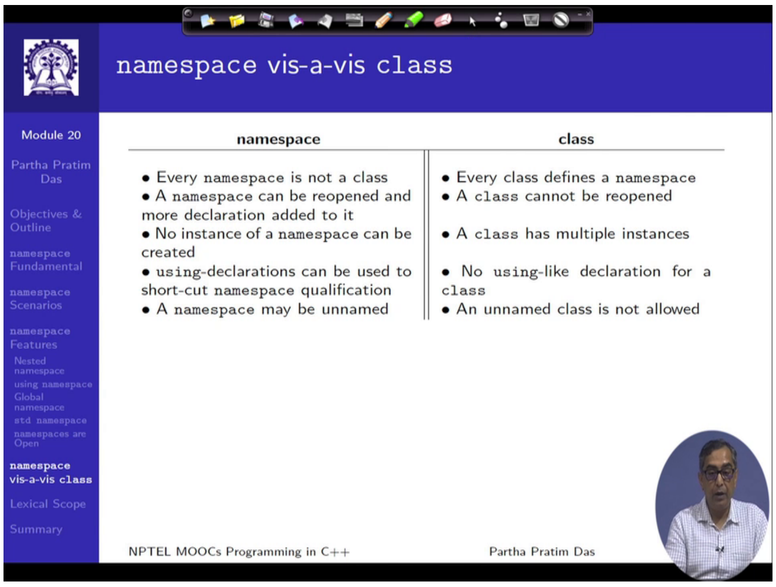
Program 20.08: namespaces are Open

- namespace are open: New Declarations can be added

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

namespace open
{ int x = 30; }

namespace open
{ int y = 40; }

int main() {
    using namespace open;
    x = y = 20;
    cout << x << "  " << y ;
    return 0 ;
}

Output: 20  20
```

One very interesting concept about namespace is namespaces are open, in the sense that; think about a class if you just define a class, then whatever symbols you put within that class, the data members, the functions and so on, the friend and all those that has to be put into the one integral definition of the class. And, once that scope is over then you cannot add new symbols to that scope of the class. But in namespace, that is different. So, that is what is meant by the namespaces being open. So here, I have created a namespace open, where I have put a symbol x. The scope is closed; this started here, it closed it here. But then, I again say namespace open and put another symbol; that is... So, what happens is, this will get added to the same scope. Now, it says basically, the namespace open has two symbols, symbol x as well as symbol y. So, simply if we say using namespace open then I can use both x and y and from the same namespace, they will, x will bind here and y will bind here. So, this openness is an interesting concept which is very flexible, so that, you can, in multiple, different files also, different parts of the namespace may be specified.

(Refer Slide Time: 25:27)



namespace vis-a-vis class

namespace	class
• Every namespace is not a class	• Every class defines a namespace
• A namespace can be reopened and more declaration added to it	• A class cannot be reopened
• No instance of a namespace can be created	• A class has multiple instances
• using-declarations can be used to short-cut namespace qualification	• No using-like declaration for a class
• A namespace may be unnamed	• An unnamed class is not allowed

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

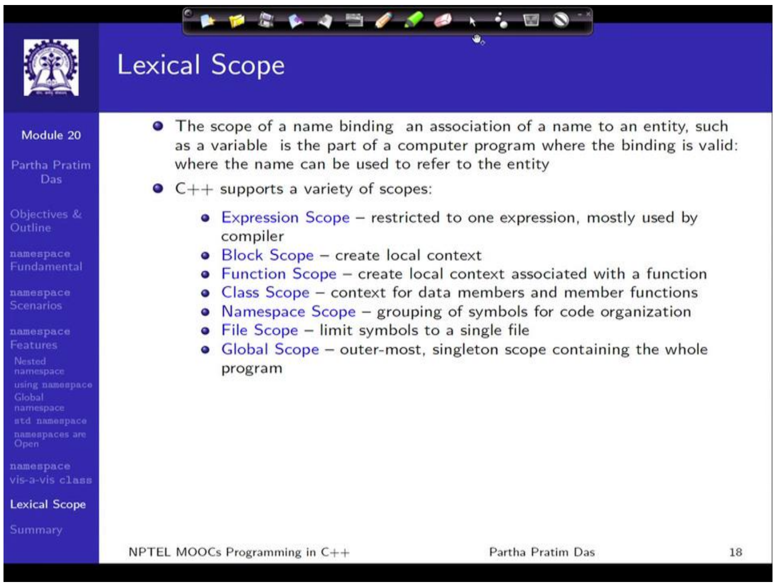
Lexical Scope

Summary

NPTEL MOOCs Programming in C++ Partha Pratim Das

Since we have been talking, referring frequently to the concept of class and comparing with the namespace this is just a summary of that comparison. Between the namespace and the class every namespace is not a class and every class in turn defines a namespace. it does give you the same qualification ability. Namespaces can be reopened, but and more declarations put into it. In terms of class, there is nothing like that. Certainly, namespaces cannot be instantiated; classes are meant to be instantiated for objects. Using mechanism is available for namespace; certainly, for class there is no sense of doing that. And interestingly, a namespace may be unnamed; I can have a namespace just to segregate some of the symbols and just to put them together, but not to access them from outside. See, why you need the names of the namespace so that from outside, by using the using declaration or directly you can access the symbols inside the namespace. Now, if I just want to blindly hide some symbols and just want them to interact between themselves I can put them in a namespace and not give a name to that namespace. If I do not give a name to that namespace, nobody from outside that namespace has an access to the symbols within that namespace. So, unnamed namespaces have a meaning; obviously unnamed class is meaningless. It is not allowed.

(Refer Slide Time: 26:56)



Lexical Scope

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

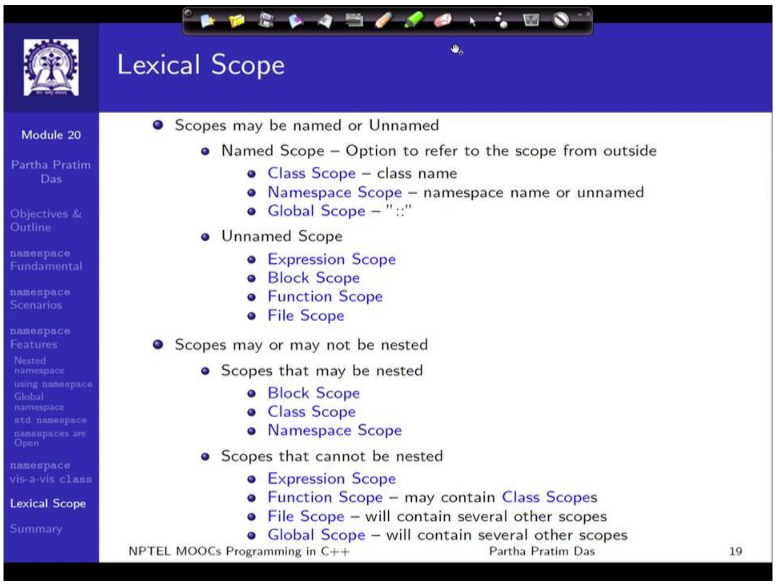
Summary

- The scope of a name binding an association of a name to an entity, such as a variable is the part of a computer program where the binding is valid: where the name can be used to refer to the entity
- C++ supports a variety of scopes:
    - Expression Scope – restricted to one expression, mostly used by compiler
    - Block Scope – create local context
    - Function Scope – create local context associated with a function
    - Class Scope – context for data members and member functions
    - Namespace Scope – grouping of symbols for code organization
    - File Scope – limit symbols to a single file
    - Global Scope – outer-most, singleton scope containing the whole program

NPTEL MOOCs Programming in C++ Partha Pratim Das 18

Before I close I would like to just remind you that, namespace belongs to one of the different lexical classes that C++ define. And, this is just to recap that these are the different lexical scopes that you have. The expression scope is what you, where the temporaries are used for computing different parts of an expression and they have a scope within the expression itself. Most often, we do not get to see these temporaries. So, it is only compiler who handles that. But, we have been frequently dealing with the block and function scope, particularly in C; and, also the file and global scope in C. And, having come to C++, these exist and in addition we have the class scope and the namespace scope that we have just discussed.

(Refer Slide Time: 27:51)



Lexical Scope

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

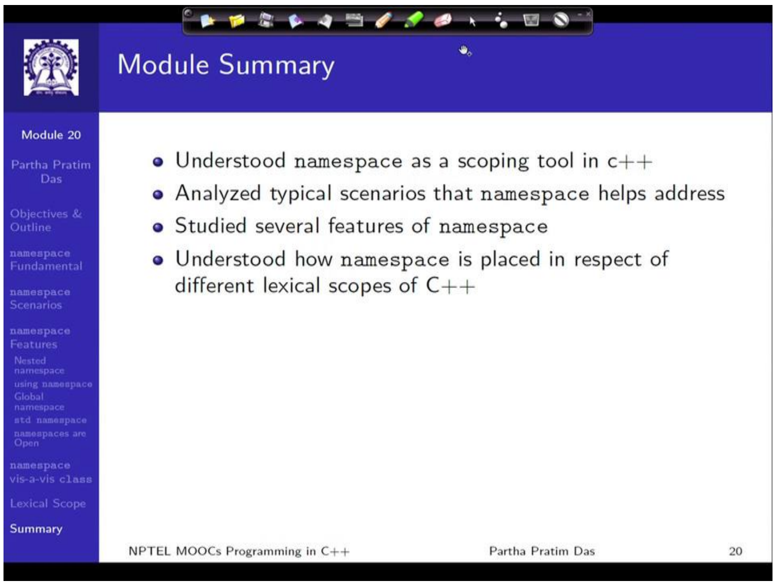
- Scopes may be named or Unnamed
  - Named Scope – Option to refer to the scope from outside
    - Class Scope – class name
    - Namespace Scope – namespace name or unnamed
    - Global Scope – "::"
  - Unnamed Scope
    - Expression Scope
    - Block Scope
    - Function Scope
    - File Scope
- Scopes may or may not be nested
  - Scopes that may be nested
    - Block Scope
    - Class Scope
    - Namespace Scope
  - Scopes that cannot be nested
    - Expression Scope
    - Function Scope – may contain Class Scopes
    - File Scope – will contain several other scopes
    - Global Scope – will contain several other scopes

NPTEL MOOCs Programming in C++ Partha Pratim Das 19

As you can note that scopes may be named or unnamed; like class scope will always have to have a name. Namespace scope will usually have a name, but may be unnamed also. Global scopes do not have a name, but can be identified by this scope resolution operator. And these scopes, like expression, block, function and file scope, do not have any names they are unnamed. And also, the scopes may be nested, like block scope, class scope, namespace scope; they may be nested. But, there are certain scopes like function scope or expression scope, they cannot be nested. But some of them can contain other scopes, but may not contain themselves.

So, this is just to summarise because it is very important to, specifically know the lexical scope, because C++ happens to be a strongly lexically scoped language. It does not have dynamic scoping; that is, it does not have execution dependent name binding. It is a whole name binding, that is what it associates, how it associates memory with a variable name is completely dependent on the lexical scope, completely dependent on the static time. Therefore, it is very important that out of these all different options of specifying variable names and restricting their visibility and accesses, which is the right one, in a right design situation and use that.

(Refer Slide Time: 29:24)



Module Summary

Module 20

Partha Pratim Das

Objectives & Outline

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Summary

- Understood namespace as a scoping tool in c++
- Analyzed typical scenarios that namespace helps address
- Studied several features of namespace
- Understood how namespace is placed in respect of different lexical scopes of C++

NPTEL MOOCs Programming in C++ Partha Pratim Das 20

And, namespace only helps to extend that kitty of lexical scopes and particularly, is a powerful tool for organising your code and particularly accessing libraries and segregating your library from symbols of your library from symbols of other third party libraries and so on. For example, if you are developing a library which you want to share out to others, it always a good idea that you put a meaningful namespace name to that whole thing and put that whole development within that namespace, like the standard library is using std and then give it to the user, so that it will not have a possibility that the, you have used certain function names or class names in your library which the user also wants to use, but are not able to do so.

So, that is about the namespaces and we will close here.