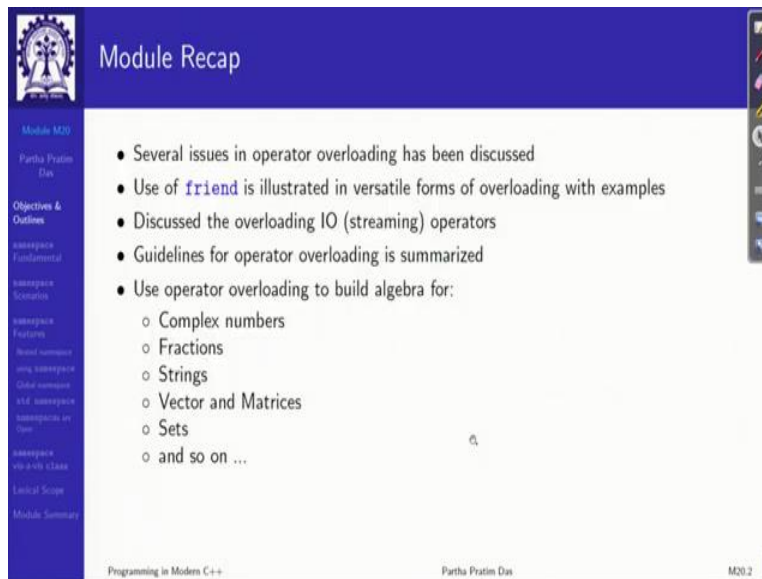


Programming in Modern C++
Professor. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture - 20
Namespace

Welcome to Programming in Modern C++ we are in week 4. And I am going to discuss module 20.

(Refer Slide Time: 00:38)



The screenshot shows a presentation slide titled "Module Recap" with a blue header. On the left, there is a vertical navigation menu with the following items: "Module M20", "Partha Pratim Das", "Objectives & Outlines", "Namespace Fundamental", "Namespace Scenarios", "Namespace Features", "Build Summary", "Unit Assessment", "Self Assessment", "Self Assessment on Code", "Assessment via a Web Class", "Lecture Scope", and "Module Summary". The main content area contains a bulleted list of topics discussed in the module. At the bottom of the slide, it says "Programming in Modern C++", "Partha Pratim Das", and "M20.2".

- Several issues in operator overloading has been discussed
- Use of `friend` is illustrated in versatile forms of overloading with examples
- Discussed the overloading IO (streaming) operators
- Guidelines for operator overloading is summarized
- Use operator overloading to build algebra for:
 - Complex numbers
 - Fractions
 - Strings
 - Vector and Matrices
 - Sets
 - and so on ...

In the last module, we concluded our discussions on operator overloading, looking at how to overload operators using global function, member function as well as friend function and how it can be used to build more complete algebra of user defined types.

(Refer Slide Time: 00:59)

The slide is titled "Module Objectives" and features a blue header with a logo on the left and a small video feed of the presenter on the right. The main content area is white and contains a single bullet point: "• Understand `namespace` as a free scoping mechanism to organize code better". A vertical sidebar on the left lists various topics under "Module M20" and "Objectives & Outlines". The footer includes "Programming in Modern C++", "Partha Pratim Das", and "M20.3".

In the present module will take up another concept which enhances the readability and expressability of C++ provides a lot of engineering benefits, it is called namespace.

(Refer Slide Time: 01:12)

The slide is titled "Module Outline" and features a blue header with a logo on the left and a small video feed of the presenter on the right. The main content area is white and contains a numbered list of six items: "1 namespace Fundamental", "2 namespace Scenarios", "3 namespace Features" (with sub-bullets: "• Nested namespace", "• using namespace", "• Global namespace", "• std namespace", "• namespaces are Open"), "4 namespace vis-à-vis class", "5 Lexical Scope", and "6 Module Summary". A vertical sidebar on the left lists various topics under "Module M20" and "Objectives & Outlines". The footer includes "Programming in Modern C++", "Partha Pratim Das", and "M20.4".

So, these are this is an outline of things to expect.

(Refer Slide Time: 01:17)

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
- Oblivates the use of File Level Scoping of C (file) static

Hand-drawn diagram:

file1.c: `int x;`
`static int y;`

file2.c: `extern int x;`
`int z;`

Arrows indicate that file2.c depends on file1.c for the definition of 'x'.

Programming in Modern C++ | Partha Pratim Das | M20 6

So, let me define what is namespace? Namespace is a declarative region that provides a scope to identify. You are familiar with the scope, you know function scope, block scope, global scope. So, you know that every time you put a pair of curly braces, you are creating a scope. If it is associated with the function, you have a function scope. If it is not inside any function, it is in the global scope. If it is associated with a class, then it is class scope and so on.

Namespace provides a class like modularization. But without the semantics of the class, which creates a it is a very strong paradigm, where the fine level scoping of C static can be avoided. In in C program file, and therefore, in C++ program file as well, you can declare a variable to be static or a function to be static, what it means? That this is local, so, if I have two files, say, file1.c, I have another file2.c and I have int x here, then I can use this extern int x in a different file.

So, what happens is this is meant to be a global variable, which can be referenced here. If I do not want that to happen, then I can write it as static say int y, I can write int y in the other. Now, what it means that this static int y is available only within this file, it is not available here, this is not same as the global int y that you have.

So, this is called a file local scope, which is very useful because when two developers are developing into different, why was this given in C is the basic factors, when two developers are developing, at the same time, they are writing into different files. And it is quite likely that they

will use the same name. Now, if they use the same name, but they are not actually the same variable, if they are actually the same variable, then one except everything else will be externed and linker will put them together.

But if they are not the same name, so, if I in this I have int z and int this two I have int z, then it is necessary a linkage error because you are defining two different global variables by the same name. This is which is not permitted. So, (file the) static in the file scope was provided in C to take care of such situations, which is eliminated by the use of namespace, we will see how.

(Refer Slide Time: 04:34)

Program 20.01: namespace Fundamental

- 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();
}

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

Programming in Modern C++ Partha Pratim Das M20.7

Program 20.01: namespace Fundamental

- 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();
}

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

Programming in Modern C++ Partha Pratim Das M20.7

Program 20.01: namespace Fundamental

Module M20
Partha Pratim Das

Objectives & Outlines
namespace Fundamental

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();
}

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

Programming in Modern C++ Partha Pratim Das M20.7

So, let us first look at a program. The namespace is defined with the keyword namespace. And then, the name of the namespace. So, you can see that the syntax is very similar to struct or class, except for the keyword. So, what I have is in this namespace, MyNameSpace, I have met different declarations. I have declared a data, I have declared a function, I have also declared a class. Now, all of these names are scoped inside the name of the namespace.

What it means is if, if I had declared say somewhere here, if I had declared `int myData`, then how do you refer to `myData`? By `myData`, it is global. But in this case, the name of `myData` is qualified by the namespace name. That is called a namespace scope. So, when I say `MyNameSpace::myData` it means this particular variable. When I want to talk about this function, I write it as `MyNameSpace::myFunction`.

Name of this class is `MyNameSpace::MyClass`. So, it is exactly like the class scoping that we have already seen the name scoping in the class of qualified names in the class. But namespace is just a grouping, it has no other behavior, like data member, public, private, instance, it is just a lexical concept, it is just a syntactic concept. It just allows you to resolve your name structured your names in a nice way. So, that is a that is a way you use the namespace.

(Refer Slide Time: 06:40)

Scenario 1: Redefining a Library Function Program 20.02

- `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: <code>abs()</code>	namespace: <code>abs()</code>
<pre>#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 }</pre>	<pre>#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 }</pre>

Programming in Modern C++ M20.9

Scenario 1: Redefining a Library Function Program 20.02

- `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: <code>abs()</code>	namespace: <code>abs()</code>
<pre>#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 }</pre>	<pre>#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 }</pre>

Programming in Modern C++ M20.9

Scenario 1: Redefining a Library Function
Program 20.02

- 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: <code>abs()</code>	namespace: <code>abs()</code>
<pre>#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 }</pre>	<pre>#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 }</pre>

So, what are the different scenarios? Let us say, I mean I will just illustrate you some of the pitfalls where namespace can really help. Let us say, talking about the C standard library, `stdlib`, there is an `abs()` function, which takes the absolute, which gives you the absolute value of an integer. Now, suppose you want to create a specific `abs()` function, a special `abs()` function, so that it will return the absolute value of parameter `n`, if `n` is between `-128` and `127`.

Otherwise, it should return 0. So, it is like it is like, so it is 0, it is 0, and in between only. If it is positive, it remains positive. If it is negative, it remains, you get positive between `-127` to `128`. You want to do this, you want to write this function. Now, if you have to do that, then you will write a function like this. If `n` is less than `-128`, return 0 greater than `127` return 0, otherwise return `-n`.

It is not difficult to write that function. The question is what do you call the function? The moment you give it the name `abs()` and have the same signature, what it does it hides, it will hide the original function in the `cstdlib` library. So, now, whenever you use `abs()`, it will use your function only, this is what you get. So, there are two aspects, one is if you do not want to use a different name for `abs()`, because that is not semantically expressive, you cannot, you do not want to call it `myabs()`, that can always solve the problem.

But you want to have, call it `abs`, but provide it with a different definition. You cannot do that without hiding the original function that exists in the library. So, this is, this is the basic problem

of name clash. So, you have a name in the global already `int abc()` in terms of. And you want to create another identity for that name in your space. So, that is when the namespace can help.

What you do is you do not clutter the global space, you write your function, but put that into your own namespace, say `myNS`. If you put it in your namespace then what is the name of this `abs()`? The name of this `abs()` is qualified by this. So, when you want to use your function, you call it with the namespace qualifier. When you call it independently, separately without the namespace, just call it as `abs()`, you still get the global function that exists in the `cstdlib`. You can see the differences in the output to understand this.

(Refer Slide Time: 10:17)

Scenario 1: Redefining a Library Function
Program 20.02

- `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: <code>abs()</code>	namespace: <code>abs()</code>
<pre>#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 }</pre>	<pre>#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 }</pre>

Programming in Modern C++ | Partha Pratim Das | M20 9

We are doing it for `-203`, `-677`, `179`. So, these two are within the range of `-128` to `127` and these two are outside. So, here you get `0`, `0`. And here you get the absolute value. Whereas, if you use just `abs()` which is from `cstdlib` in every case, you will just get the corresponding positive. So, this is one place where namespace can be really really useful.

(Refer Slide Time: 10:51)

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>
#include <cstring>
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(St* s) { rec[nStudents] = s; s->setRoll(++nStudents); }
    St *getStudent(int r) { return (r == nStudents + 1) ? 0 : rec[r - 1]; }
};
```

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

Programming in Modern C++ Partha Pratim Das M20.10

Let us look at a different problem, not I should not say different, but an extension of this problem. Think about an organization that is developing an application to process student records. So, class St is for students and class StReg (register) is a list of students. So, here, we try to define the class St and class StReg (register). It has a typedef, for enumerated gender. It has a constructor, which constructs a student record with the gender.

It is, it can set roll number, it can get the gender, and it has an output operator, you have just learned how to do that. And it has the name, gender and roll as its private data members. Similarly, in the registry, all that you need is a list of students. So, it is array of these pointers. And you want to know what is number of students and you can have constructor, adding a student and getting details about the student. This is, this has got nothing, nothing to do with namespace, I am just trying to create the context for a near real situation.

(Refer Slide Time: 12:29)

Scenario 2: Students' Record Application: Team at Wc Program 20.03

- 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 ProcSt() { 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("Ravi", St::male); reg->add(&s); ProcSt(); }

Processing for females by Niloy
///////////////////////////////// App1.cpp ///////////////////////////////////
#include <iostream>
using namespace std;
#include "Students.h"
extern StReg *reg;
void ProcSt() { 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("Rhea", St::female); reg->add(&s); ProcSt(); }
```

Now, after having written this, let us say the organization has a lead engineer called Poornima. And there are two developers by the name of Sabita and Niloy. So, Poornima writes these classes, your St class and StReg class, the header file says student.h. And then allocates the task, divide the task of implementation to Sabita and Niloy. So, both are given student.h header to work with.

And Sabita is told that please write all the code that are specifically applies to the male students. Niloy is told write all the code that specifically applies to the female students. So, I mean, what those codes can be what differs, what, their requirements and all that we are not getting into all those. So, because that is not relevant for us. The point is that both of them will work with the same set of classes and write separate applications for certain subgroups of objects.

Sabita for male student object, Niloy for the female student object. Then just to model this processing function is say it is just printing, it is for male students, it is printing the student record if it is a male student here, the female students and female recordings. And corresponding to this they create application where they test it taking Ravi as an instance and gender as male, and check that the registration and processing are happening right.

So, does Niloy, she creates Rhea whose gender is female registers and process this. So, but once Poornima has given them the input to them is the processing requirement. And in terms of code,

this student.h header file which has the classes. Otherwise, they are working independently, mostly.

(Refer Slide Time: 14:51)

Scenario 2: Students' Record Application: Integration I
Program 20.03

- 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 ProcSt(); // Function from App1.cpp by Sabita ✓
void ProcSt(); // Function from App2.cpp by Niloy ✓

StReg *reg = new StReg(1000);

int main() {
    St s1("Rhea", St::female); reg->add(&s1); ✓
    St s2("Ravi", St::male); reg->add(&s2); ✓
    ProcSt(); // Function from App1.cpp by Sabita ✓
    ProcSt(); // Function from App2.cpp by Niloy ✓
}
```

- But the integration failed due to name clash
- Both use the same signature `void ProcSt()`; 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

Programming in Modern C++ Partha Pratim Das M20.12

Now, after they are done, then they give their code back to Poornima. And who has to integrate, because finally, the application has to work for both male students as well as female students. And she has to integrate both into a single main function, in our main function. So, she also includes student.h, she includes the processing function header from Sabita, as well as Niloy includes the instances that they have created makes a call as the date.

But this integration as you can see will fail because of the name clash. Because what happened is Niloy and Sabita, both have chosen to use the same function name. Therefore, when Poornima has to integrate code from them, then naturally there are two `ProcSt()` functions in the same project, which cannot exist. And you do not know which one is being called. If you, if you have this, you do not know that it is from App1 by Sabita or this from App2 by Niloy.

So, this becomes now a real nightmare. Now I am just trying to highlight the problem at a very, very miniscule level and pathological level. But in a even in a medium sized project with 5, 6, 10 developers working on different parts, such type of name clash is very, very likely to happen, because everybody has a, has a kind of inherent tendency to put natural names. So, if I am working on electronic circuits, I will call an adder as an adder.

And my friend Shamir who is working on other types of adder will also call an adder as an adder and the function names global names all will start clashing. So, this has happened, umpteenth time in the industry that because of such now, if it could be planned and designed thoroughly that well, this is the name you use, this is the name that you will use. Often that is not the case, often that is not if Poornima had to do so much, then possibly she would have written the code itself.

She has created the skeleton, the classes, given to Sabhita given to Niloy told them to do details, and they come back having a number of name clashes.

(Refer Slide Time: 17:51)

Scenario 2: Students' Record Application: Wrap in namespace Program 20.03

- 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 ProcSt() {
        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 ProcSt() {
        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;
    }
};
    
```

Programming in Modern C++ Partha Pratim Das M20.13

So, the way to one way to very quickly resolve this is to basically put these into different namespaces. So, what Poornima does, she puts a namespace App1 for Sabita's code and namespace App2 for Niloy's code puts everything there. So, what now happened what happens? The ProcSt which was ProcSt and ProcSt was clashing in the name. Now this becomes App1::ProcSt, this becomes App2::ProcSt, they become different.

So once you set the namespace, then you can either work as Sabita or work as Niloy. If we just change the namespace, then it is a different code space that we are working in, without actually having to change the code, edit the code that are embedded inside.

(Refer Slide Time: 18:44)

Scenario 2: Students' Record Application: A Good Night Program 20.03

- Now the integration gets smooth:
using namespace std;

```
#include "Students.h"

namespace App1 { void ProcSt(); } // App1.cpp by Sabita
namespace App2 { void ProcSt(); } // App2.cpp by Niloy

StReg *reg = new StReg(1000);

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

    App1::ProcSt(); // App1.cpp by Sabita
    App2::ProcSt(); // App2.cpp by Niloy
    return 0;
}
```

- Clashing names are made distinguishable by distinct namespaces

Programming in Modern C++ Partha Pratim Das M20.14

So, all that Poornima does, she puts them in the different namespaces and qualifies the name with this, the clash is resolved. So, that is a, this is one of the very big advantages of having namespace. Besides, there are several others but this is one of the big advantages that you can do quick engineering, of merging code spaces from different without bothering about the name clash.

And I just talked about to developers doing this, it could also happen that you have a lot of developed code and you decide to integrate with a third party library, that third party library has global symbols, you have global symbols and there is a clash. So, it is always a preferred practice in C++ that when you work for your particular sub project, sub module or for yourself, you do everything inside a namespace, which is unique for you so that there is no possibility of clash with other global names.

(Refer Slide Time: 19:55)

namespace Features

namespace Features

Programming in Modern C++ Partha Pratim Das M20.15

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;
}
```

Handwritten annotation: name1::name2::data

Programming in Modern C++ Partha Pratim Das M20.15

So, what are the features of namespaces? So, just a quick set of details. In namespace may be nested within another namespace. So, I have a namespace name 1, this is something which is outside the namespace. So, it is global. I have one namespace and I have another namespace within that namespace. So, what it happens is basically the scope resolution gets nested. So, this data, the data, which is global, is called data, no namespace.

The data which is just in named one will be named1::data, that is a, now what is this data? This is name within name2, but name2 itself is in name1. So, the actual name of name2 is name1::name2. Therefore, the data they are in is that ::data. So, this is basically given in this way. So

you can nest it in in any form. So, very simple logic, all intuitive things that you get in terms of this.

(Refer Slide Time: 21:18)

Program 20.05: Using using namespace and using for shortcut

- 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
}
```

Programming in Modern C++ Partha Pratim Das M20.17

Program 20.05: Using using namespace and using for shortcut

- 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
}
```

Programming in Modern C++ Partha Pratim Das M20.17

Program 20.05: Using using namespace and using f

- 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
}
```

Programming in Modern C++ Partha Pratim Das M20.17

Program 20.05: Using using namespace and using f

- 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
}
```

Programming in Modern C++ Partha Pratim Das M20.17

Now often what happens as we saw name1::name2::data, you have to write long prefixes. So, there is a shortcut feature given, which is through another keyword using and we say using namespace, you must have seen this line couple of hundred times by now, but what does that actually mean?

What it means is, this is a shortcut to say that if I have a namespace name1, having two symbols, and another namespace name2 having two symbols not nested, I can specify by saying using namespace and then give a name of a namespace or a specific symbol. So, when I just write this, using namespace name1, then subsequently whatever I write, the symbols will be looked inside that namespace.

So, for example, for name1, then I have written v11, but v11 actually is name1::v11. But I did not have to write that, because I have said using name1. So, when I do v11 it knows that it will look for the scope of name1, and it will access this. I can do v21, I have, I am sorry, v12 after this, I can, it is not necessary that I will have to call it v12, I can always also use the fully qualified name.

I could have called it as v12 as well. But I have written it as (v1) name1::v12. Similarly in the other, I have said that using name2::v21 that is here that using is not on a namespace. So, I have not said namespace, I have just said using this, which means that it is a symbol inside some namespace. So then, so it is name2::v2 that is this one. So, when you say v2, it actually means this.

I do not have to say anything else. But and I can do fully qualified also. But mind you see the difference that if I put v22 which is in name2, I will get an undefined error. Why? But for v12, I do not get I will not get I mean I can I can use simply v12 and that will also be perfect. Why is this difference? Because in case of name1, I have put a using on namespace overall. So, all symbols in that are in the available list.

But here, I have just specifically said that I am going to use v21 from name2. So, I am not saying that I will use v22. So, it will if I have to use it I have to specifically say name2::v22. Now naturally, this way it, I mean a lot of writing can be shorter, but it might lead to other problems also. For example, if I had say, a say suppose I had a symbol v21 here also, then with this, I will have a problem.

Because what happens if I talk about v21. The fact that my name1 namespace is default, it means the v21 here. But I have said that name2::v21 must be default, but that is a different variable. So, I have to make sure that by this using I do not again by using what I am doing I am again shortcutting the names, so I am creating possibilities for clash again. So, this kind of using will have to be avoided.

(Refer Slide Time: 25:49)



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
}
```

• Items in Global namespace may be accessed by scope resolution operator (::)

Now the question is what happens with the global names? There is a data here, there is a data here. And I am saying using name1::data, which means that if I say data, it means the data of name1. So, when I say data now it means data of name1. If I say name1::data, it also means data of name1. How do I talk about the data in the global. So, when this using I am hiding the global data?

The global namespace does not have a name. So, there is a special syntax given where you do not have a name, but you just say ::, which means a global namespace. So, if I want to refer to the global data, I just write it as ::data. So, writing data will mean this one because of this using. But writing ::data will mean the global one, that is how you can access all the namespaces appropriately.

(Refer Slide Time: 27:02)

Program 20.07: std Namespace

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

Without using using std	With using using std
<pre>#include <iostream> ✓ int main() { int num; std::cout << "Enter a value: "; std::cin >> num; std::cout << "value is: "; std::cin << num; }</pre>	<pre>#include <iostream> ✓ using namespace std; ✓ int main() { int num; cout << "Enter a value: "; cin >> num; cout << "value is: "; cin << num; }</pre>

- 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
- 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

Programming in Modern C++ Partha Pratim Das M20.19

So, now, you can understand the std namespace we have been talking about. So, all C++ standard library components are put under one single namespace std. So, that allows, that leaves the global namespace completely free for the user, you can put anything there. Though it is advised that you do not put anything you put in your own namespace, but you are free to do that. So, if I have to access symbols from this, so I have to typically write it as std::, only then it will work.

So, when I put this using namespace std, I mean that these symbols should be looked into this namespace. So, std::cout can now be written simply as cout, that is a basic difference. So, that is the reason we always keep on doing this. And if we do not do that, then we would have to put the fully qualified name in the std namespace where all standard library symbols belong.

(Refer Slide Time: 28:20)

Program 20.08: namespaces are Open

- namespace are open: New Declarations can be added

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

namespace open // First definition
{int x = 30; }

namespace open // Additions to the last definition
{int y = 40; }

int main() {
    using namespace open; // Both x and y would be available
    x = y = 20;
    cout << x << " " << y ;
}

Output: 20 20
```

Programming in Modern C++ Partha Pratim Das M20.20

One very nice thing about the namespaces they are open, what it means that you can define a few symbols in the namespace. For example, I am doing here in a namespace called open, then I can again say namespace open and add symbols to that. This is not something that you can do in a class. So, it is not that whole of the namespace has to be created together.

So, for example, all of the standard libraries put in the std namespace, but if I want I can get say namespace std and add five symbols to that. So, this is a so what happens is when you when this when you say using namespace open, it means this as well as this as if all of them have been taken union off, as if they were declared at the same point but they were actually not. And then x and y will refer to the respective one you can figure it out from the output that they are resolved and getting accessed properly.

(Refer Slide Time: 29:22)

The slide is titled "namespace vis-a-vis class" and is presented in a blue-themed interface. It features a table comparing "namespace" and "class".

namespace	class
<ul style="list-style-type: none">• Every namespace is not a class• A namespace can be reopened and more declaration added to it• No instance of a namespace can be created• using-declarations can be used to short-cut namespace qualification• A namespace may be unnamed	<ul style="list-style-type: none">• Every class defines a namespace• A class cannot be reopened• A class has multiple instances• No using-like declaration for a class• An unnamed class is not allowed


Handwritten notes in blue ink are present on the slide: "namespace" with a curly brace underneath it, and a closing curly brace "}" below that.

At the bottom of the slide, the text "Programming in Modern C++" is on the left, "Partha Pratim Das" is in the center, and "M20.22" is on the right.

Comparing a namespace with the class, very easy in every namespace is not a class but every class defines a namespace, we have been talking about this. Namespaces can be reopened and more declarations can be added, classes cannot be reopened. No instance of a namespace can be created. It has no sense, class has multiple instances. using namespace is a shorthand for name qualification.

No such thing exists for a class, even when you are going to say defining the static variables and so on. And namespace maybe unnamed also. That is I may just say, I may just say that is not putting anything here, yes, there is. We just guard safe from any other namespace or global, but does not allow you to reopen, because there is no name, it can be done only once. So, these are some of the specific things that happen.

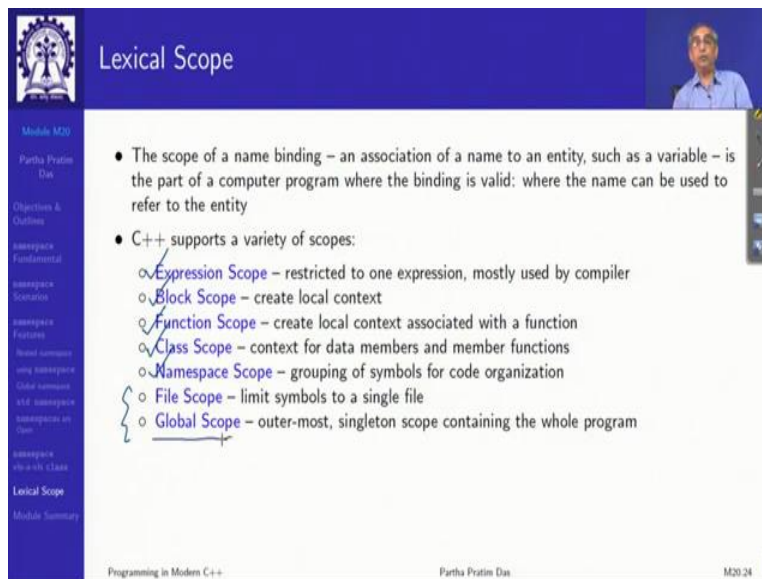
(Refer Slide Time: 30:27)



The slide is titled "Lexical Scope" in white text on a blue header. The main content area is white with the words "Lexical Scope" written in red. A sidebar on the left contains a navigation menu with items like "Module M20", "Partha Pratin Das", "Objectives & Outlines", "namespace Fundamental", "namespace Scopes", "namespace Features", "Name namespace", "using namespace", "Global namespace", "std namespace", "namespace in file", "namespace in a class", "Lexical Scope", and "Module Summary". The footer includes "Programming in Modern C++", "Partha Pratin Das", and "M20.23". A small video feed of the presenter is in the top right corner.

So, with the introduction of the namespace, now, if you look at we have a wide variety of lexical scope that is available.

(Refer Slide Time: 30:34)



The slide is titled "Lexical Scope" in white text on a blue header. The main content area is white with a bulleted list of scope types. A sidebar on the left is identical to the previous slide. The footer includes "Programming in Modern C++", "Partha Pratin Das", and "M20.24". A small video feed of the presenter is in the top right corner.

- 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

I started talking about that we have lexical scoping in C++, its scope is within which a particular name has a binding, has a meaning is available as a variable or a function or class etc. So, you have a scope for the expression, which is restricted to only one expression. You have a scope for the block, which is the local context, you know what is a block. You have a scope for the function, you have a scope for the class.

We just learned the scope for the namespace and you have a file scope, where I explained the meaning of the word static for a, for a in the fine scope, which is again, not something that you should be using any more. And you have the outermost global scope, which again, you should not be using except for putting your classes. So, these are your different lexical scopes that are available in a C++ program.

(Refer Slide Time: 31:36)

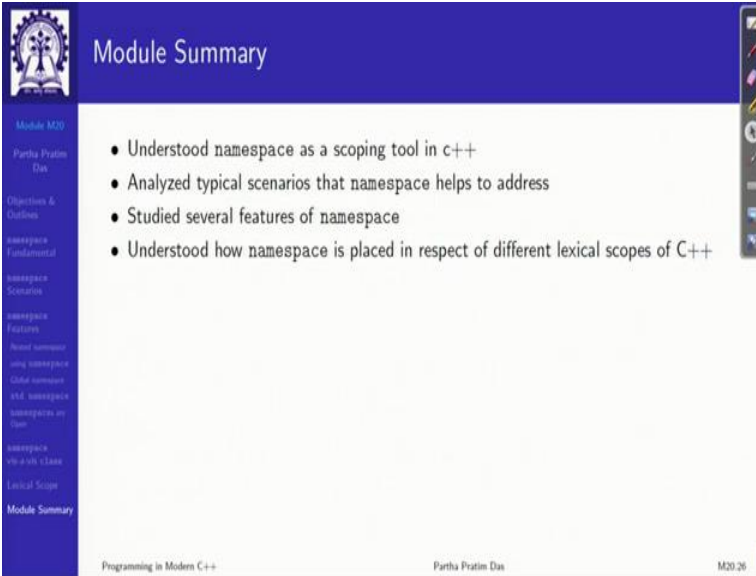
Lexical Scope

- 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

Programming in Modern C++ Partha Pratim Das M20.25

And scopes may be named or unnamed. So, named scopes or class scope named scope with the name global when qualified with the ::. Unnamed scopes are expression scope, there is no name given. Block scope, blocks cannot be given name. Function scope, no separate name same as a function. The file scope again, there is no separate name and they may be nested or not nested. Block, class namespace this can be nested, but expression, function, file or global scope cannot be nested. So, this is an overall scenario in terms of the scoping in of different symbols in C++.

(Refer Slide Time: 32:22)



Module Summary

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

Programming in Modern C++ Partha Pratim Das M20.26

So, we have understood and talked about various aspects of namespace as a scoping tool in C++, which is of great use and importance in terms of organizing the symbols, particularly at a, at a global or near global level. And we have studied different features for that. Thank you very much. I hope you have enjoyed this discussion. And be very careful as you start your engineering projects.

Be careful to define your namespace, define the namespace of your sub module properly, and then work on it so that you will not be having nightmares with name clashes. Thank you very much.