



## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace  
Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

# Module 20: Programming in C++ Namespaces

Instructors: Abir Das and Jibesh Patra

Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur

*{ abir, jibesh }@cse.iitkgp.ac.in*

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**



# Module Objectives

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

### Objectives & Outlines

`namespace`  
Fundamental

`namespace`  
Scenarios

`namespace`  
Features

Nested `namespace`

using `namespace`

Global `namespace`

`std namespace`

`namespaces` are  
Open

`namespace`  
`vis-a-vis class`

Lexical Scope

Module Summary

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



# Module Outline

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

### Objectives & Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace  
Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- 1 namespace Fundamental
- 2 namespace Scenarios
- 3 namespace Features
  - Nested namespace
  - using namespace
  - Global namespace
  - std namespace
  - namespaces are Open
- 4 namespace vis-a-vis class
- 5 Lexical Scope
- 6 Module Summary



# namespace Fundamental

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- 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
- A [good resource](#) for File Level Scoping in C Something Linky



# Program 20.01: namespace Fundamental

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace  
std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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();
}
```

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



# Scenario 1: Redefining a Library Function

## Program 20.02

### Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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
            }
```

### 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
            }
```



# Scenario 2: Students' Record Application: The Setting Program 20.03

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- 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`



# Scenario 2: Students' Record Application: Team at Work

## Program 20.03

### Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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 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**

```
////////// App2.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(); }
```





# Scenario 2: Students' Record Application: Integration Nightmare: Program 20.03

## Module 20

Instructors: Abir Das and Jibesh Patra

Objectives & Outlines

namespace Fundamental

namespace Scenarios

namespace Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are Open

namespace vis-a-vis class

Lexical Scope

Module 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 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 clashes**
- **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`**



# Scenario 2: Students' Record Application: Wrap in namespace

## Program 20.03

### Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace  
Global namespace

std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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 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;
    }
};
```



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

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace  
Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- 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 names



# Program 20.04: Nested namespace

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- 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;
}
```



# Program 20.05: Using using namespace and using for shortcut

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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
}
```



# Program 20.06: Global namespace

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- `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 (`::`)



# Program 20.07: std Namespace

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace  
**std namespace**  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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 ;
}
```

- 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 ;
}
```

- 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



# Program 20.08: namespaces are Open

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

- 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





# namespace vis-a-vis class

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace

using namespace

Global namespace

std namespace

namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module Summary

## namespace

- 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

## class

- 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



# Lexical Scope

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace

std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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



# Lexical Scope

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace

Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

Module 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



# Module Summary

## Module 20

Instructors: Abir  
Das and Jibesh  
Patra

Objectives &  
Outlines

namespace  
Fundamental

namespace  
Scenarios

namespace  
Features

Nested namespace  
using namespace  
Global namespace  
std namespace  
namespaces are  
Open

namespace  
vis-a-vis class

Lexical Scope

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++