Namespaces

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Namespaces

- A namespace is a collection of name definitions, such as class, variable, and function
  - If a program uses classes and functions written by different programmers, it may be that the same name is used for different things
  - Namespaces help us deal with this problem

The Using Directive

- <iostream> header file places names such as **cin** and **cout** in the **std** namespace
- The program does not know about these names in the **std** namespace until you add

  using namespace std;

(If you do not use the std namespace, you can define cin and cout of your own to behave differently)

The Global Namespace

- Code you write goes in a namespace
  - it is in the **global namespace** unless you specify a namespace
  - The global namespace **does not require** the **using** directive

Name Conflicts

- If the same name is used in two namespaces, the namespaces cannot be used at the same time
  - Example: **my_function** is defined in namespaces **ns1** and **ns2**, the two versions of **my_function** could be used in one program by using **local** using directives this way

```
{
  using namespace ns1;
  my_function( );
}
```

```
{
  using namespace ns2;
  my_function( );
}
```

Scope Rules For using

- A block is a list of statements enclosed in { }
- The scope of a using directive is the block in which it appears
- A using directive placed at the beginning of a file, outside any block, applies to the entire file

Creating a Namespace

- To place code in a namespace, use a namespace grouping.

```
namespace NameSpaceName
{
        SomeCode
}
```

- To use the namespace created, use the appropriate using directive.

```
using namespace NameSpaceName;
```

Namespaces: Declaring a Function

- To add a function to a namespace, declare the function in a namespace grouping

```
namespace apollo
{
        void greeting( );
}
```

Namespaces: Defining a Function

- To define a function declared in a namespace, define the function in a namespace grouping.

```
 namespace apollo
  {
     void greeting( )
       {
                    cout << "Hello from namespace savitch1.\n";
       }
   }
```

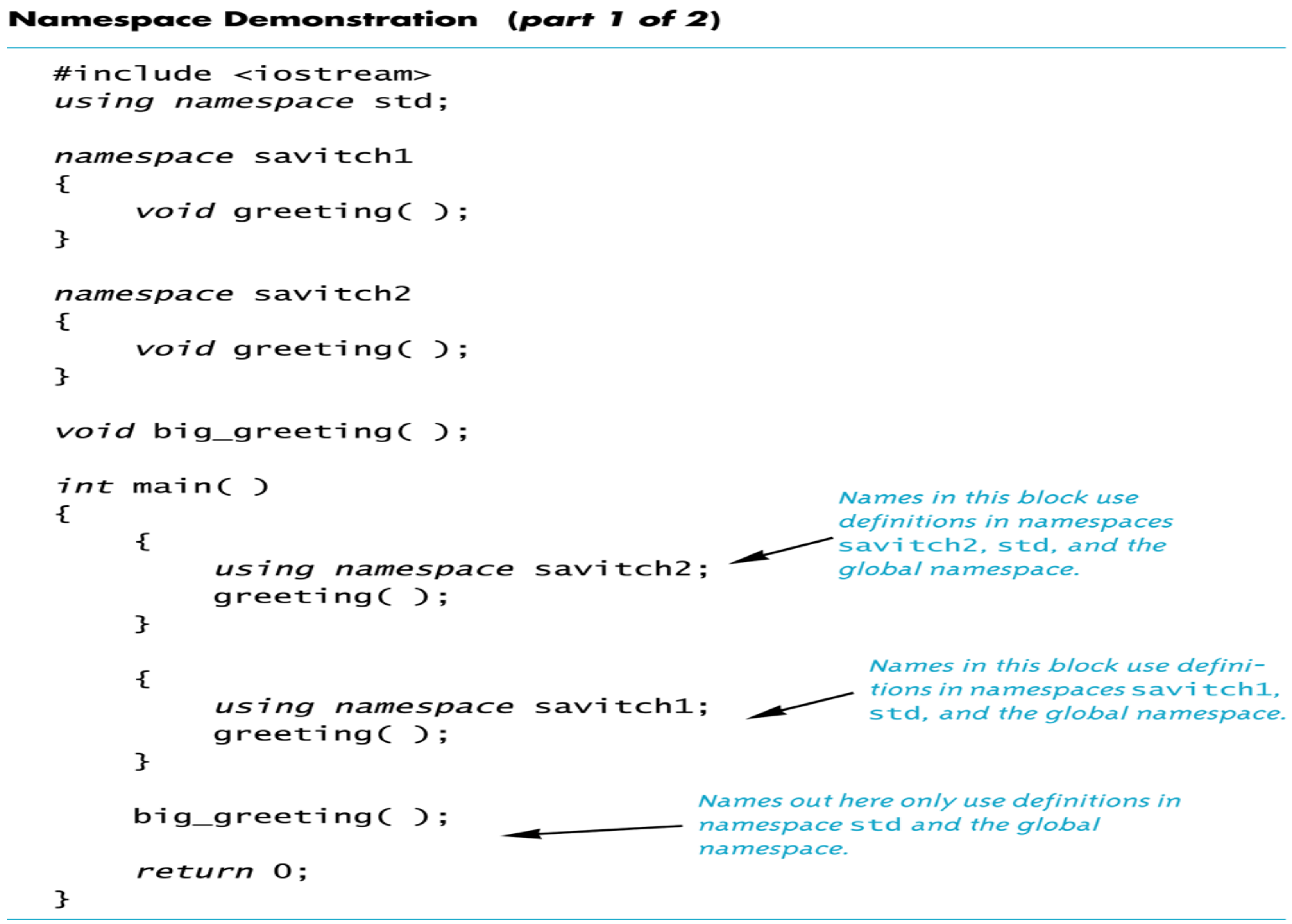
Namespaces: Using a Function

- To use a function defined in a namespace
  - Include the using directive in the program where the namespace is to be used
  - Call the function as the function would normally be called

```
int main( )  {
    using namespace apollo;
    greeting( );
}
```

using directive's scope

Namespaces: Using a Function



Namespace Demonstration ***(part 1 of 2)***

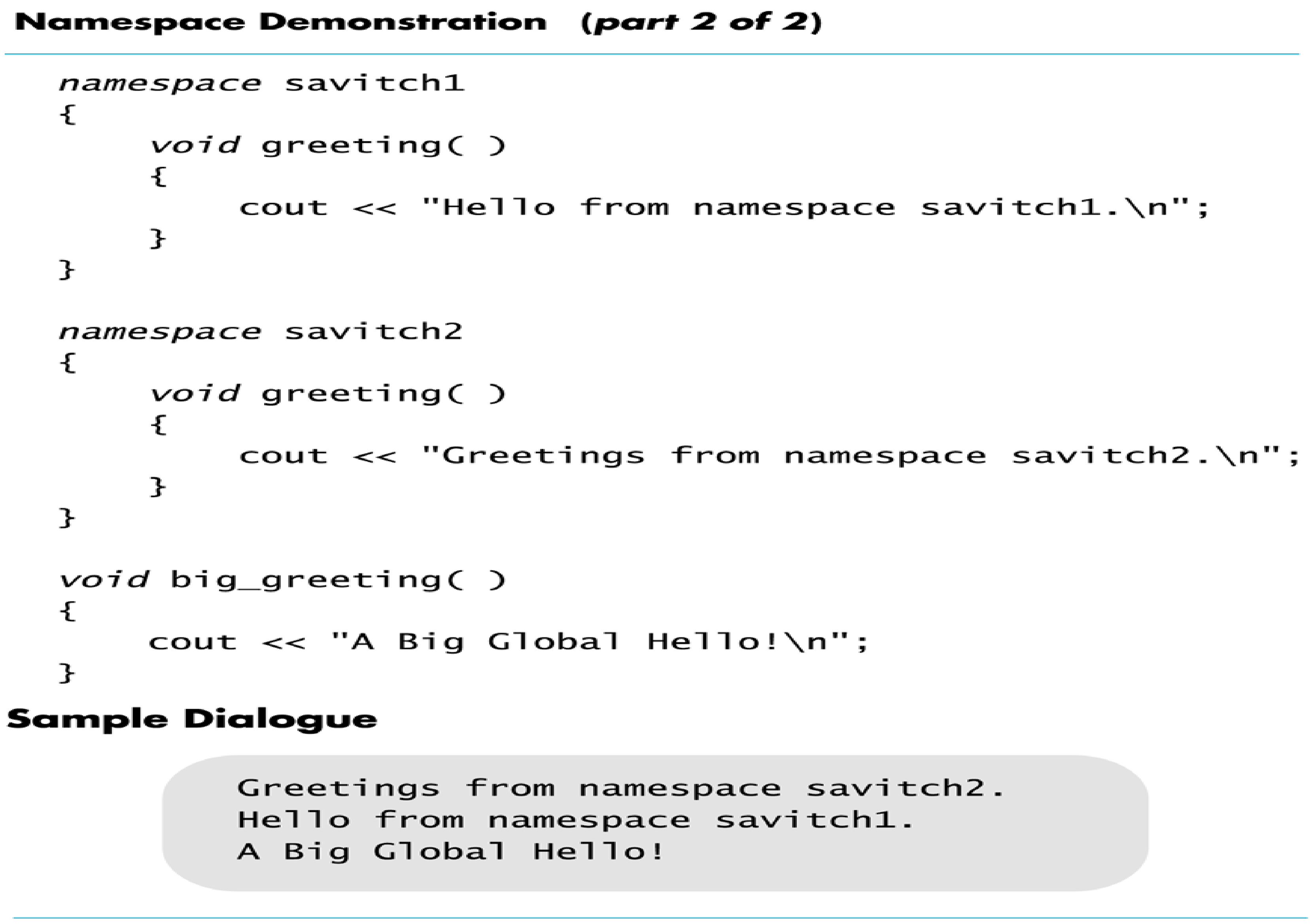
```
#include <iostream>
using namespace std;

namespace savitch1
{
     void greeting( );
}

namespace savitch2
{
     void greeting( );
}

void big_greeting( );

int main( )
{
     {
          using namespace savitch2;
          greeting( );
     }

     {
          using namespace savitch1;
          greeting( );
     }

     big_greeting( );

     return 0;
}
```

Names in this block use definitions in namespaces `savitch2, std,` *and the global namespace.*

Names in this block use definitions in namespaces `savitch1, std,` *and the global namespace.*

Names out here only use definitions in namespace `std` *and the global namespace.*

Namespaces: Using a Function



Namespace Demonstration ***(part 2 of 2)***

```
namespace savitch1
{
    void greeting( )
    {
        cout << "Hello from namespace savitch1.\n";
    }
}

namespace savitch2
{
    void greeting( )
    {
        cout << "Greetings from namespace savitch2.\n";
    }
}

void big_greeting( )
{
    cout << "A Big Global Hello!\n";
}
```

Sample Dialogue

```
Greetings from namespace savitch2.
Hello from namespace savitch1.
A Big Global Hello!
```

A Namespace Problem

- Suppose you have the namespaces below:

```
namespace ns1
{
      fun1( );
      my_function( );
}
```

```
namespace ns2
{
      fun2( );
      my_function( );
}
```

- Is there an easier way to use both namespaces considering that **my_function** is in both?

Qualifying Names

- Using declarations (not directives) allow us to select individual functions to use from namespaces

```
using ns1::fun1;
```

  - makes only **fun1** in **ns1** available
  - The scope resolution operator identifies a namespace here means we are using only namespace **ns1's** version of **fun1**

- If you only want to use the function once, call it like this

```
ns1::fun1( );
```

Qualifying Parameter Names

- To qualify the **type** of a **parameter** with a using declaration, use the namespace and the type name.

```
int get_number (std::istream input_stream) {

}
```

  - istream is defined in namespace std
  - If istream is the only name needed from namespace std, then **you do not need to use**

    using namespace std;

Directive/Declaration

- A **using declaration** (using std::cout;) makes only one name available from the namespace
- A **using directive** (using namespace std;) makes all the names in the namespace available
- A using directive potentially introduces a name
- If **ns1** and **ns2** both define **my_function**,

```
using namespace ns1;
using namespace ns2;
```

  is **OK**, provided **my_function** is **never used**!

A Subtle Point

- A using declaration introduces a name into your code: no other use of the name can be made

```
using ns1::my_function;
using ns2::my_function;
```

is **illegal**, even if **my_function** is never used

Unnamed Namespaces

- The **unnamed namespace** can hide **helper functions**
  - Names defined in the unnamed namespace are **local** to the **compilation unit**
  - A compilation unit is a file (such as an implementation file) plus any file(s) #included in the file

The unnamed grouping

- Every compilation unit has an unnamed namespace
- The namespace grouping is written as any other namespace, but no name is given:

```
 namespace  {
     void sample_function( );
      …
}  //unnamed namespace
```

Names in the unnamed namespace

- **Names** in the **unnamed namespace**
  - Can be reused outside the compilation unit
  - Can be used in the compilation unit without a namespace qualifier

Compilation Units Overlap

- If a header file is included in two files
    - It is in two compilation units
    - Participates in two unnamed namespaces!
    - This is OK as long as each of the compilation units makes sense independent of the other
        - A name in the header file's unnamed namespace cannot be defined again in the unnamed namespace of the implementation or application file

Global or Unnamed?

- Names in the global namespace have global scope (all files)
  - They are available without a qualifier to all the program files
- Names in the unnamed namespace are local to a compilation unit
  - They are available without a qualifier within the compilation unit