C/C++ Programming Language

CS219 Spring

Feng Zheng

Lecture 8

SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Content

- Brief Review

- Separate Compilation

- Storage Duration, Scope, and Linkage
  - ➢ Scope and Linkage
  - ➢ 1. Automatic Storage Duration
  - ➢ 2. Static Duration Variables: External, Internal and No Linkage
  - ➢ 3. Storage Schemes and Dynamic Allocation

- Namespaces

- Summary

Brief Review

Content of Last Class

- **Adventures in functions**
  - ➢ Inline function
  - ➢ Reference variables
  - ➢ Default arguments
  - ➢ Function overloading
  - ➢ Function template

Separate Compilation

Separate Compilation

- C++ allows to locate the component functions of a program in separate files
  - ➢ Modify one file and then recompile just that one file
  - ➢ Make it easier to manage large programs
  - ➢ Most IDEs provide additional facilities to help with the management (The make programs in Unix and Linux systems)
- Divide the original program into three parts
  - ➢ A header file that contains the structure (type) declarations and prototypes for functions that use those structures (types)
  - ➢ A source code file that contains the code that define the structure (type) - related functions
  - ➢ A source code file that contains the code that calls the structure (type) - related functions

A Header File

- **Shouldn't** put function **definitions** or **variable declarations** into a header file

- Commonly found in header files
  - ➢ Function **prototypes**
  - ➢ Symbolic **constants** defined using #define or const (special linkage)
  - ➢ **Structure** declarations (not variable)
  - ➢ **Class** declarations
  - ➢ **Template** declarations (not code to be compiled)
  - ➢ **Inline** functions (special linkage)

- **Don't** add header files to the project list in IDEs

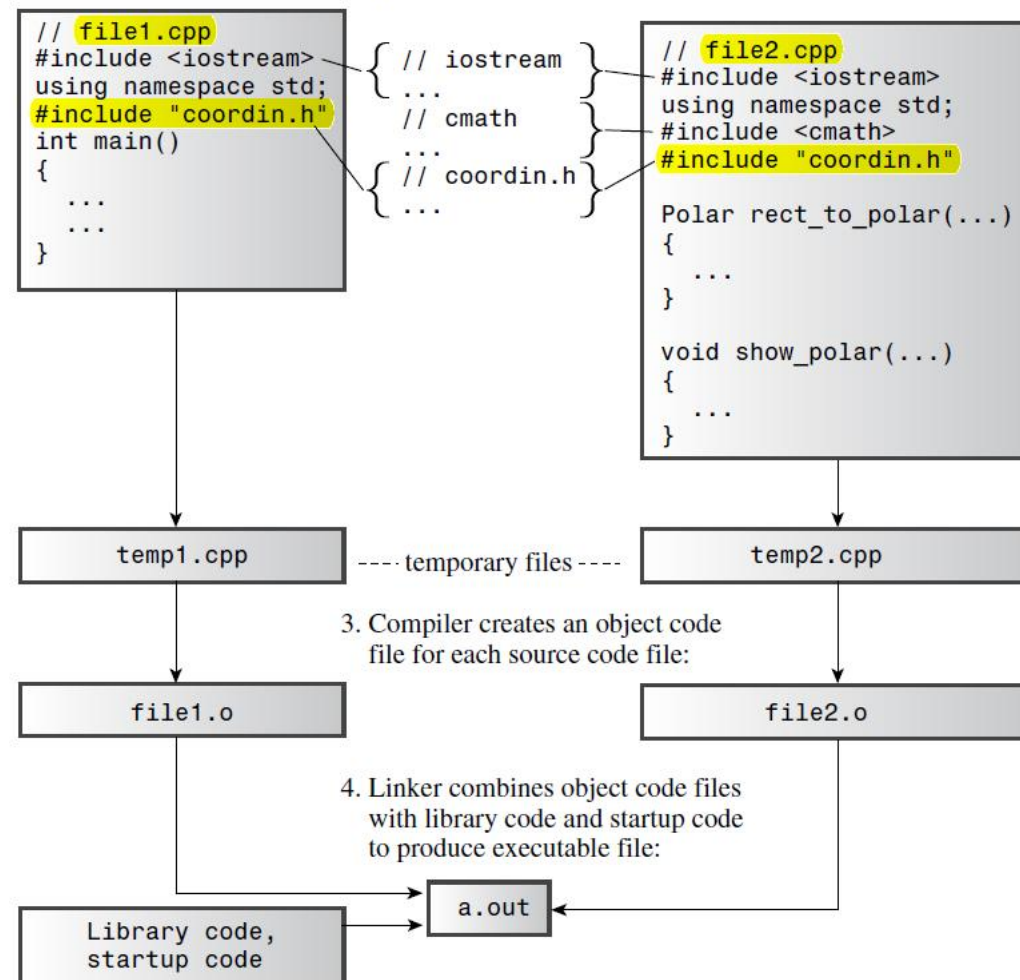- **Don't** use **#include** to include source code files in other source code files

Header File Management

- You should include a header file **just once in a file**
- Most of the standard C and C++ header files use this **guarding scheme**

```
#ifndef COORDIN_H_
#define COORDIN_H_
// place include file contents here
#endif
```

1. Give UNIX compile command for two source files:

`CC file1.cpp file2.ccp`

2. Preprocessor combines included files with source code:

```
// file1.cpp
#include <iostream>
using namespace std;
#include "coordin.h"
int main()
{

    ...
    ...

}
```

```
// iostream
...
// cmath
...
// coordin.h
...
```

```
// file2.cpp
#include <iostream>
using namespace std;
#include <cmath>
#include "coordin.h"

Polar rect_to_polar(...)
{
    ...
}

void show_polar(...)
{
    ...
}
```

| temp1.cpp | ---- temporary files ---- | temp2.cpp |

3. Compiler creates an object code file for each source code file:

| file1.o | | file2.o |

4. Linker combines object code files with library code and startup code to produce executable file:

| a.out |

| Library code, startup code |

Storage Duration, Scope, and Linkage

Storage

- **Three** plus **one** separate schemes for storing data
  - ➢ Automatic storage duration
    - ✓ Variables declared **inside** a function definition
    - ✓ They are **created** when program execution **enters** the function or block
    - ✓ The memory used for them is **freed** when execution **leaves** the function or block
  - ➢ Static storage duration
    - ✓ Using the keyword static to have static storage duration
    - ✓ Persist for the **entire time** a program is running
  - ➢ Dynamic storage duration
    - ✓ Memory allocated by the new operator persists until it is freed with the delete operator or until the program **ends**
  - ➢ Thread storage duration (C++11)
    - ✓ Allow a program to **split** computations into separate **threads**
    - ✓ Variables declared with the thread_local keyword have storage that persists for as long as the containing thread lasts

Scope and Linkage

- Scope: describe how widely **visible** a name is in a file
  - ➢ Local scope: **within** the block
  - ➢ Global (file) scope: **throughout the file after the point**
  - ➢ A function prototype scope: within the **parentheses**
  - ➢ Class scope
  - ➢ Namespace scope

- Linkage: describe how a name can be **shared** in **different units**
  - ➢ A name with **external** linkage can be shared **across files**
  - ➢ A name with **internal** linkage can be shared **within a single file**
  - ➢ Names of **automatic** variables and **static** variables within a **block** have no **linkage**

1. Automatic Storage Duration

- Function parameters and variables declared inside a function have, by default, automatic storage duration
- Have local scope and no linkage
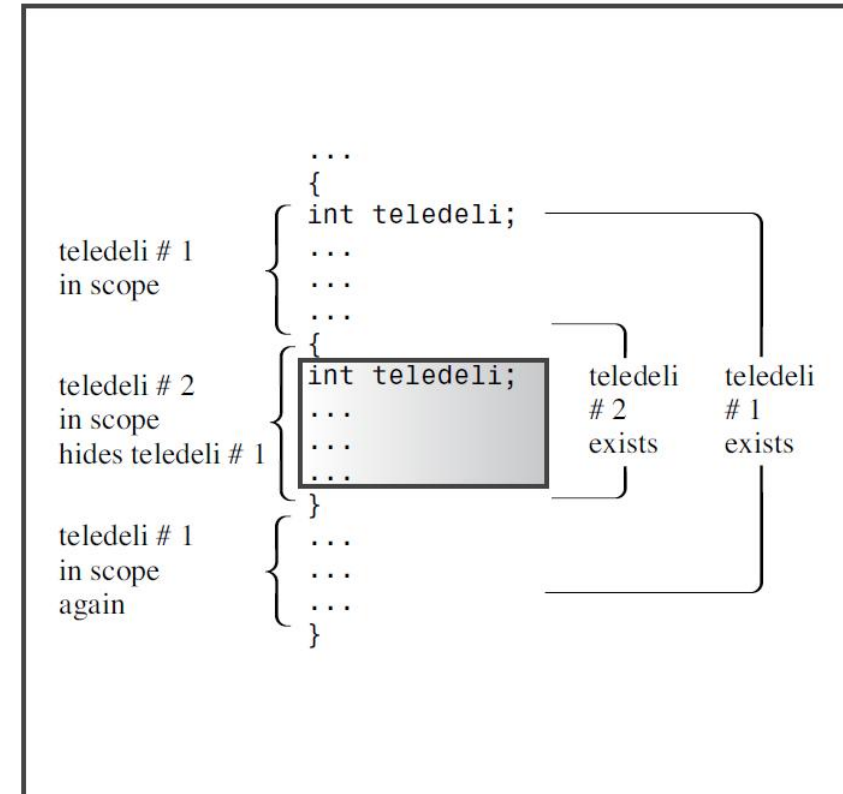
1. Two Variables: Automatic and Register

- Initialization of automatic variables
  - ➢ You can initialize an automatic variable with any expression whose value will be known when the declaration is reached

- Automatic variables and the stack (first-in-last-out)
  - ➢ Set aside a section of memory and treat it as a stack for managing the flow and ebb of variables
  - ➢ New data is figuratively stacked atop old data
  - ➢ Remove from the stack when a program is finished with it

- Register Variables
  - ➢ C originally introduced the register keyword to suggest that the compiler use a CPU register to store an automatic variable

1. Passing Arguments by Using a Stack

- Remember the address
  - ➢ Stack: first-in-last-out
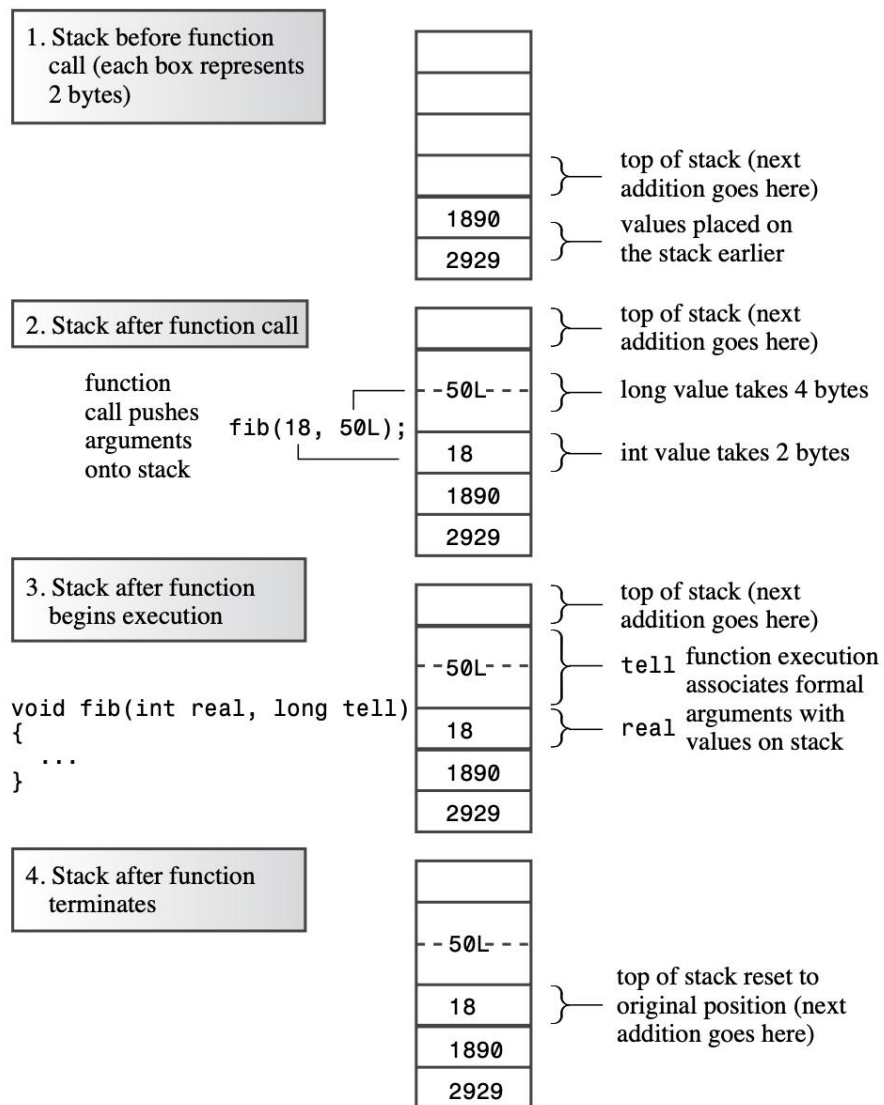
2. Static Duration Variables

- C++, like C, provides static storage duration variables with three kinds of linkage
  - ➢ External linkage (accessible across files)
  - ➢ Internal linkage (accessible to functions within a single file)
  - ➢ No linkage (accessible to just one function or to one block)

- Properties of static variables
  - ➢ The number of static variables doesn't change
  - ➢ Doesn't need a special device such as a stack to manage them
  - ➢ Allocate a fixed block of memory to hold all the static variables
  - ➢ Stay present as long as the program executes

2. Declaring and Initializing Static Variables

- Examples of declaration
- Initialization
  - ➢ Zero-initialized,
  - ➢ Constant expression initialization
  - ➢ Dynamic initialization

```cpp
#include <cmath>
int x;                              // zero-initialization
int y = 5;                          // constant-expression initialization
long z = 13 * 13;                   // constant-expression initialization
const double pi = 4.0 * atan(1.0);  // dynamic initialization
```

```cpp
...
int global = 1000;           // static duration, external linkage
static int one_file = 50;    // static duration, internal linkage
int main()
{
...
}
void funct1(int n)
{
    static int count = 0;  // static duration, no linkage
    int llama = 0;
...
}
void funct2(int q)
{
...
}
```

- What determines which form of initialization takes place?
  - ➢ All static variables are zero-initialized
  - ➢ Do simple calculations if needed

2.1 Static Duration, External Linkage

- Static variables with external linkage
  - ➢ Have static storage duration and file scope
  - ➢ Be defined outside any function
- The one definition rule
  - ➢ Have to be declared in **each** file
  - ➢ Two kinds of variable declarations
    - ✓ Defining declaration
    - ✓ Referencing declaration
  - ➢ A referencing declaration uses the keyword extern

- Run external.cpp; support.cpp

```cpp
// file1.cpp
#include <iostream>
using namespace std;

// function prototypes
#include "mystuff.h"

// defining an external variable
int process_status = 0;

void promise ();
int main()
{
    ...
}

void promise ()
{
    ...
}
```

```cpp
// file2.cpp
#include <iostream>
using namespace std;

// function prototypes
#include "mystuff.h"

// referencing an external variable
extern int process_status;

int manipulate(int n)
{
    ...
}

char * remark(char * str)
{
    ...
}
```

This file defines the variable process_status, causing the compiler to allocate space for it.

This file uses extern to instruct the program to use the variable process_status that was defined in another file.

2.2 Static Duration, Internal Linkage

- Static variables with internal linkage
  - Applying the static modifier to a file-scope variable gives it internal linkage
  - A variable with internal linkage is local to the file that contains it

  - What if you want to use the same name to denote different variables in different files?
    - Add codes (int a_int = 1;) outside any function in last program

- Run twofile1.cpp; twofile2.cpp

2.3 Static Storage Duration, No Linkage

- Static variables with no linkage

  ➢ Applying the static modifier to a variable defined inside a block
  ➢ Exist even while the block is inactive
  ➢ Preserve values between function calls
  ➢ Subsequent calls to the function don't reinitialize the variable

- Run static.cpp

Specifiers and Qualifiers

- ## Specifiers:
  - ➤ auto (eliminated as a specifier in C++11) automatic type deduction
  - ➤ register: make the data access fast
  - ➤ static: file-scope declaration and local declaration
  - ➤ extern: external linkage
  - ➤ thread_local (added by C++11)
  - ➤ mutable: a particular member of a structure (or class) can be altered even if a particular structure (or class) variable is a const

- ## CV-Qualifiers (*cv* stands for const and volatile):
  - ➤ const
  - ➤ volatile: the value in a memory location can be altered even though nothing in the program code modifies the contents (A pointer to a hardware location. Or two programs may interact, sharing data)

The Five Kinds of Variable Storage

- Comparison: summarize the storage class features as used in the pre-namespace era

Storage Description	Duration	Scope	Linkage	How Declared
Automatic	Automatic	Block	None	In a block
Register	Automatic	Block	None	In a block with the keyword `register`
Static with no linkage	Static	Block	None	In a block with the keyword `static`
Static with external linkage	Static	File	External	Outside all functions
Static with internal linkage	Static	File	Internal	Outside all functions with the keyword `static`

More Linkage

- Functions
  - ➢ C/C++ does NOT allow you to define one function inside another
  - ➢ External linkage: all **functions** automatically have static storage duration

- Function linkage
  - ➢ Use the keyword extern in a function prototype (optional)
    - ✓ Check the first program without use of extern
  - ➢ Use the keyword static to give a function internal linkage, confining its use to a single file

```
static int private(double x);
...
static int private(double x)
{
    ...
}
```

3. Storage Schemes and Dynamic Allocation

- Dynamic memory
  - ➤ Controlled by the new and delete operators
  - ➤ Not by scope and linkage rules
  - ➤ Can be allocated from one function and freed from another function
- Initialization with the new operator
  - ➤ Built-in types:
    - ✓ Using ()

```cpp
int *pi = new int (6);    // *pi set to 6
double * pd = new double (99.99);  // *pd set to 99.99
```

  - ➤ Structure or an array
    - ✓ Using {}

```cpp
struct where {double x; double y; double z;};
where * one = new where {2.5, 5.3, 7.2};  // C++11
int * ar = new int [4] {2,4,6,7};        // C++11
```

  - ➤ Remember [] ?

3. new: Operators, Functions, and Replacement Functions

- The new and new[] operators call upon two allocation functions

```
void * operator new(std::size_t);      // used by new
void * operator new[](std::size_t);    // used by new[]
```

- The placement new operator
  - ➢ Allow to specify the location to be used
  - ➢ Deal with hardware that is accessed via a particular address or to construct objects in a particular memory location

- Run newplace.cpp
  - ➢ Include the new header file
  - ➢ Use new with an argument that provides the intended address

Namespaces

Names
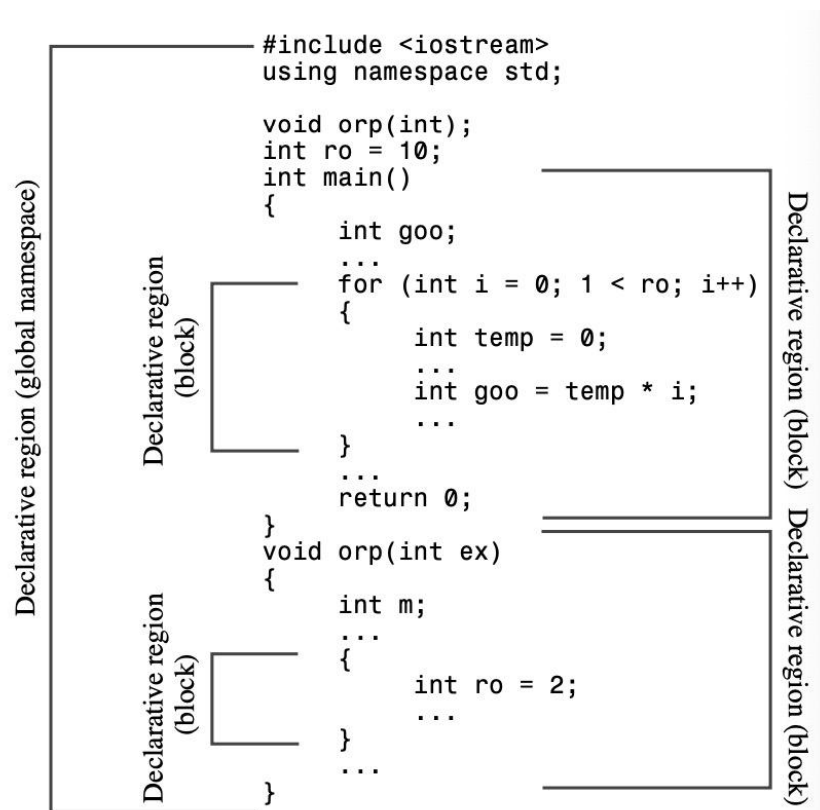
- ## Names in C++
  - ➢ Variables
  - ➢ functions
  - ➢ Structures
  - ➢ Enumerations
  - ➢ Classes
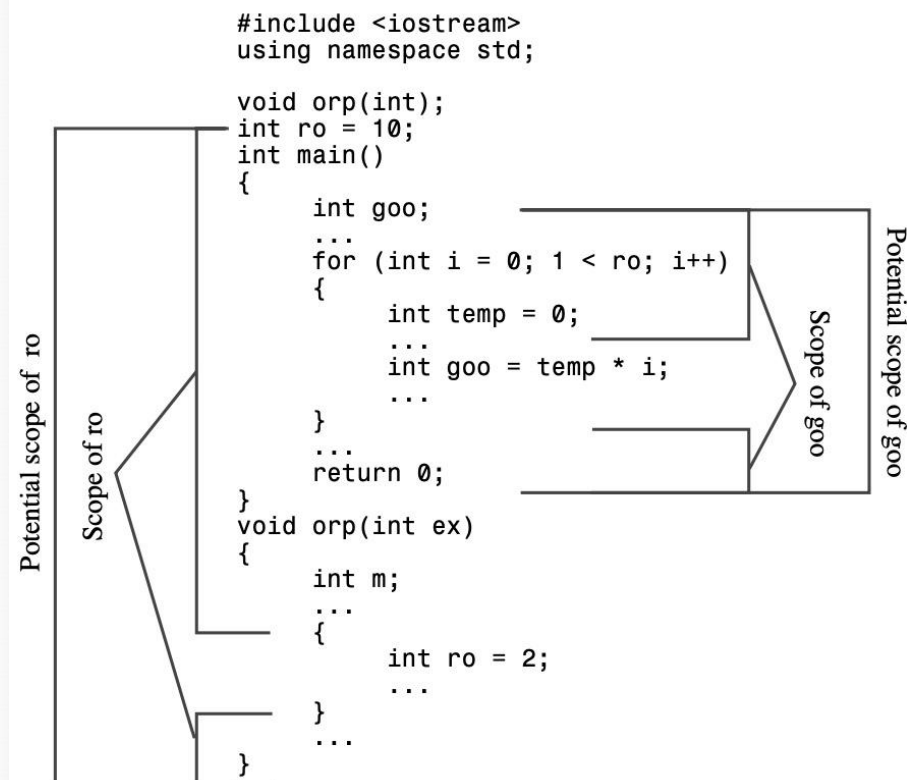 - ➢ Class and structure members

- ## Namespace problems
  - ➢ Name conflicts

Declarative regions

A region in which declarations can be made

Potential scope and scope

Begin at its point of declaration and extends to the end of its declarative region.

Namespaces

- Main purpose is to provide an area in which to declare names
  - The names in one namespace don't conflict with the same names declared in other name-spaces
  - There are mechanisms for letting other parts of a program use items declared in a namespace
  - Namespaces can be located at the global level or inside other namespaces, but they cannot be placed in a block
  - A name declared in a namespace has external linkage by default
- Global namespace
  - Correspond to the file-level declarative region,
  - Global variables are now described as being part of the global namespace

Namespaces

- using Declarations and using Directives
  - ➢ Declarations make particular identifiers available
  - ➢ Using directive makes the entire namespace accessible

```cpp
using Jill::fetch;    // a using declaration
```
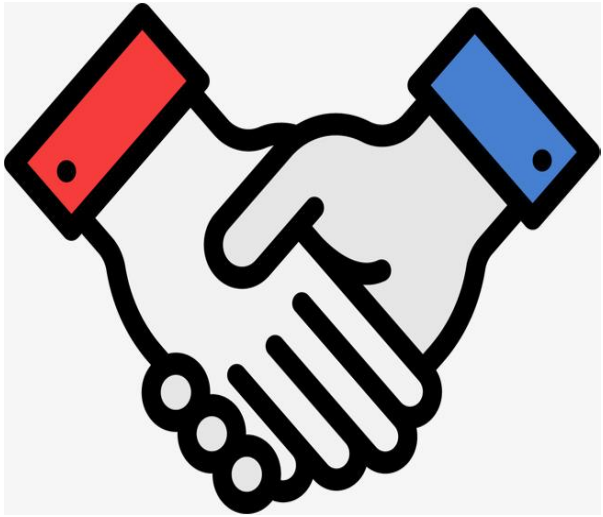
```cpp
namespace Jill {
    double bucket(double n) { ... }
    double fetch;
    struct Hill { ... };
}
char fetch;
int main()
{
    using Jill::fetch;    // put fetch into local namespace
    double fetch;         // Error! Already have a local fetch
    cin >> fetch;         // read a value into Jill::fetch
    cin >> ::fetch;       // read a value into global fetch
    ...
}
```

- Run namesp.h; namesp.cpp; usenmsp.cpp

Summary

- A header file
- Header File Management (guarding scheme)
- Scope and Linkage
  - 1. Automatic Storage Duration
  - 2. Static Duration Variables: External, Internal and No Linkage
  - 3. Storage Schemes and Dynamic Allocation
- Namespaces

Thanks

zhengf@sustech.edu.cn