Chapter 6 - Methods

6.3 Math Class Methods

6.4 Methods

6.5 Method Definitions

6.6 Argument Promotion

6.7 C# Namespaces

6.8 Value Types and Reference Types

6.9 Passing Arguments: Pass-by-Value vs. Pass-by-Reference

6.10 Random-Number Generation

6.11 Example: Game of Chance

6.12 Duration of Variables

6.13 Scope Rules

6.14 Recursion

6.15 Example Using Recursion: The Fibonacci Series

6.16 Recursion vs. Iteration

6.17 Method Overloading

Program Modules

- Modules
  - It is a small pieces of a problem, like student module, course module in the school system
  - We hope to *divide and conquer* the problem
    - Facilitate design, implementation, operation and maintenance of large programs
- Why we divide the program into modules?
  - Group working: A module can be assigned to one programmer
  - Scale down the complexity of problem
  - Isolated the naming space from other module
    - A same variable name can appear in two different modules without inference
- Interaction of modules- parameter passing
  - A big question discussed later

Program Modules

- In C, the unit of modules is function
- In **OOP language (Java, C#, …)**, the unit of modules is
  - method (a small unit), and
  - class (a large unit)
- **Note: In OOP (Java, C#, …), any method must be defined inside a class**
  - **No stand-alone method is defined outside a class in the OOP language**

Method Definitions and calling

- A method must be defined before being called
- Syntax of method definition:

```
return-value-type  method-name (parameter-list)
{
     declarations and statements
}
```

```
ex: long max (long n1, n2) {
     .... }
```

- A method, if return a value, should contain "return expression" statement, like

```
     return a+200;
```

- A method can be defined not to return a value (by void in the prefix of its definition), like

```
     void sorting(int mm[]) { .....   }
```

Method Definitions and calling

- A caller can call the method without a parameter, but the parentheses can not be omitted.

  ex: i = set( );

- With the parentheses, we can differ the method and variables

  ex: j = Math.PI * Math.Sqrt(900.0)

(Static) variable

(Static) method

Math Class Methods

- The **Math** class
  - Allows the user to perform common math calculations
  - Calculate the square root of **900.0**:
    - **Math.sqrt(900.0)**
      - Method **sqrt** (a static method) belongs to class **Math**
        - Dot operator (.) allows access to method **sqrt**
  - Constants in Math class
    - **Math.PI** = 3.1415926535…
    - **Math.E** = 2.7182818285…

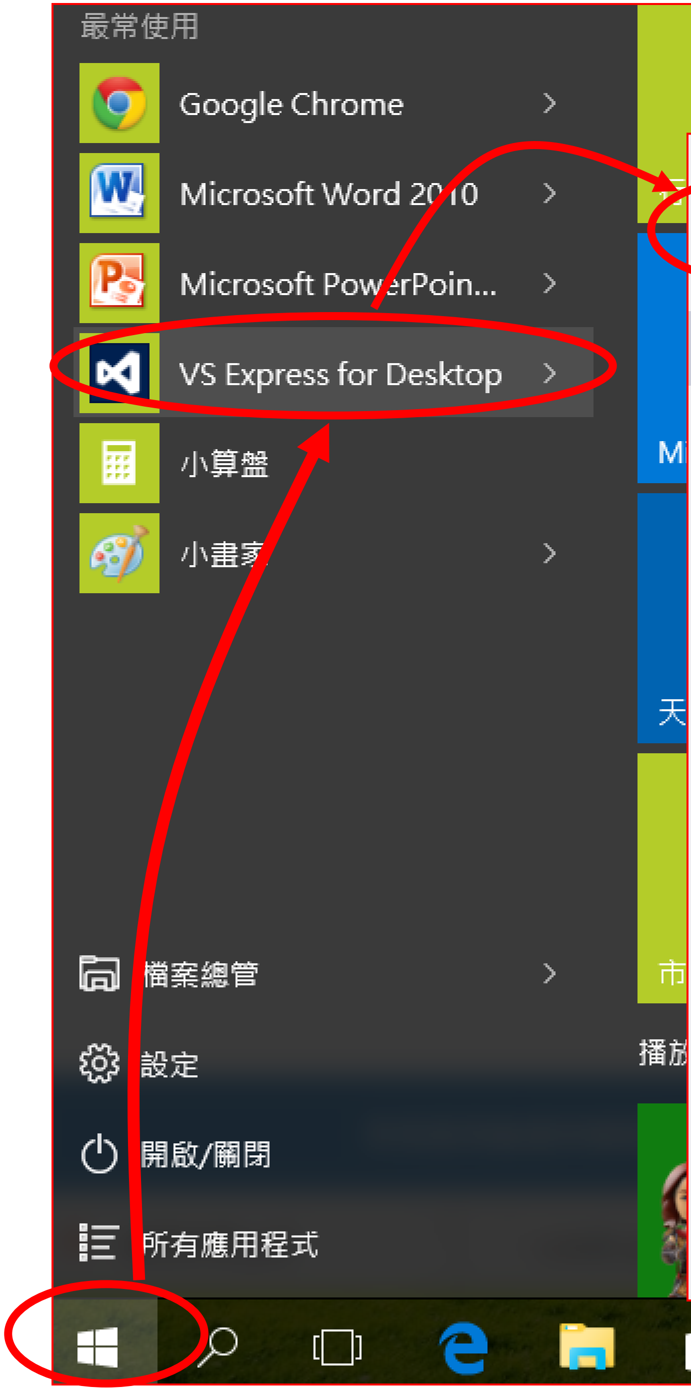
Method	Description	Example
Aabs(x)	absolute value of *x* (this method also has versions for **float**, **int** and **long** values)	**abs(23.7)** is **23.7** **abs(0.0)** is **0.0** **abs(-23.7)** is **23.7**
Ceiling(x)	rounds *x* to the smallest integer not less than *x*	**ceil(9.2)** is **10.0** **ceil(-9.8)** is **-9.0**
Cos(x)	trigonometric cosine of *x* (*x* is in radians)	**cos(0.0)** is **1.0**
Exp(x)	exponential method *ex*	**exp(1.0)** is **2.71828** **exp(2.0)** is **7.38906**
Floor(x)	rounds *x* to the largest integer not greater than *x*	**floor(9.2)** is **9.0** **floor(-9.8)** is **-10.0**
Log(x)	natural logarithm of *x* (base *e*)	**log(2.718282)** is **1.0** **log(7.389056)** is **2.0**
Max(x, y)	larger value of *x* and ***y*** (this method also has versions for **float**, **int** and **long** values)	**max(2.3, 12.7)** is **12.7** **max(-2.3, -12.7)** is **-2.3**
Min(x, y)	smaller value of *x* and ***y*** (this method also has versions for **float**, **int** and **long** values)	**min(2.3, 12.7)** is **2.3** **min(-2.3, -12.7)** is **-12.7**

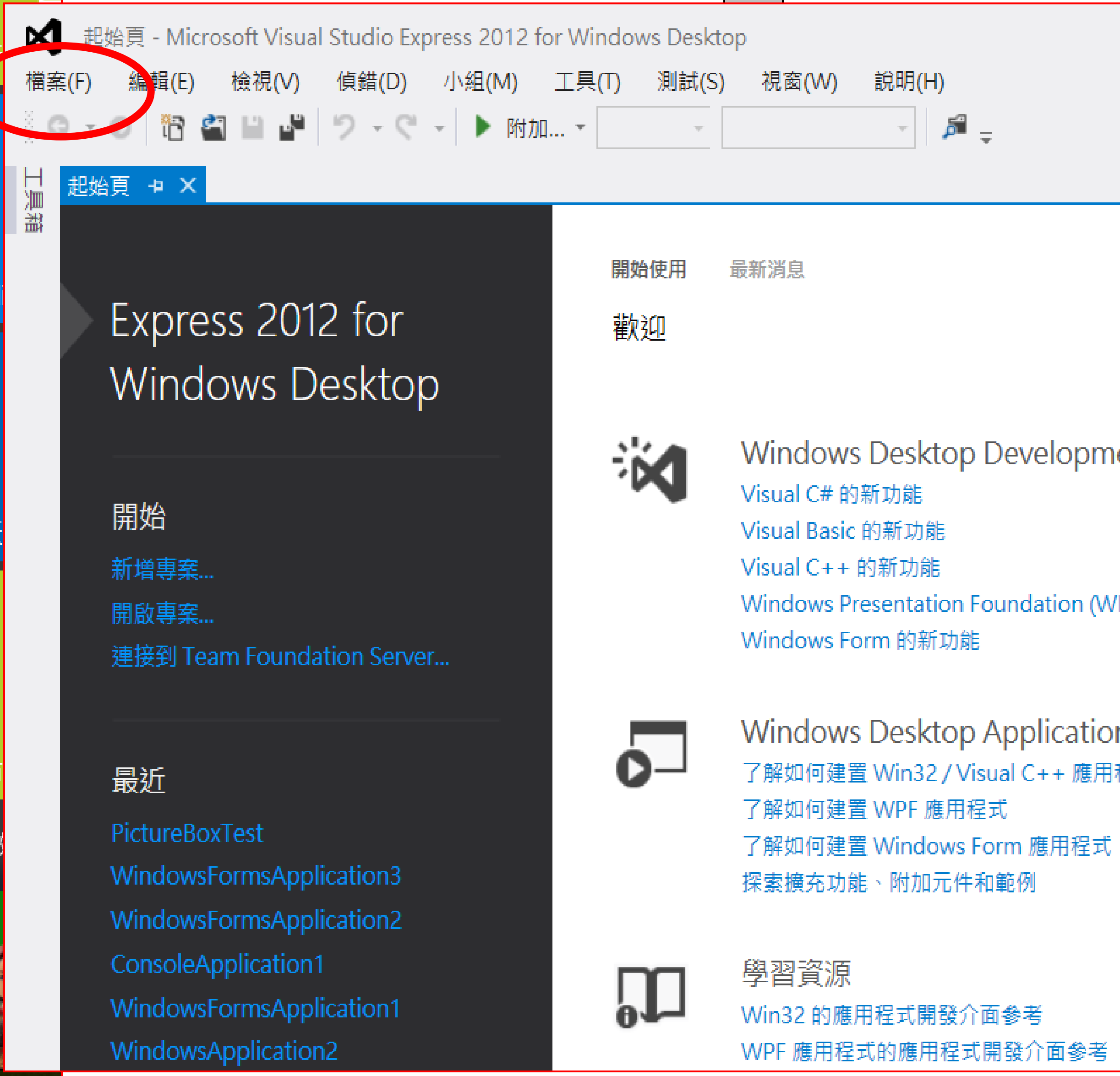
◀ ▶

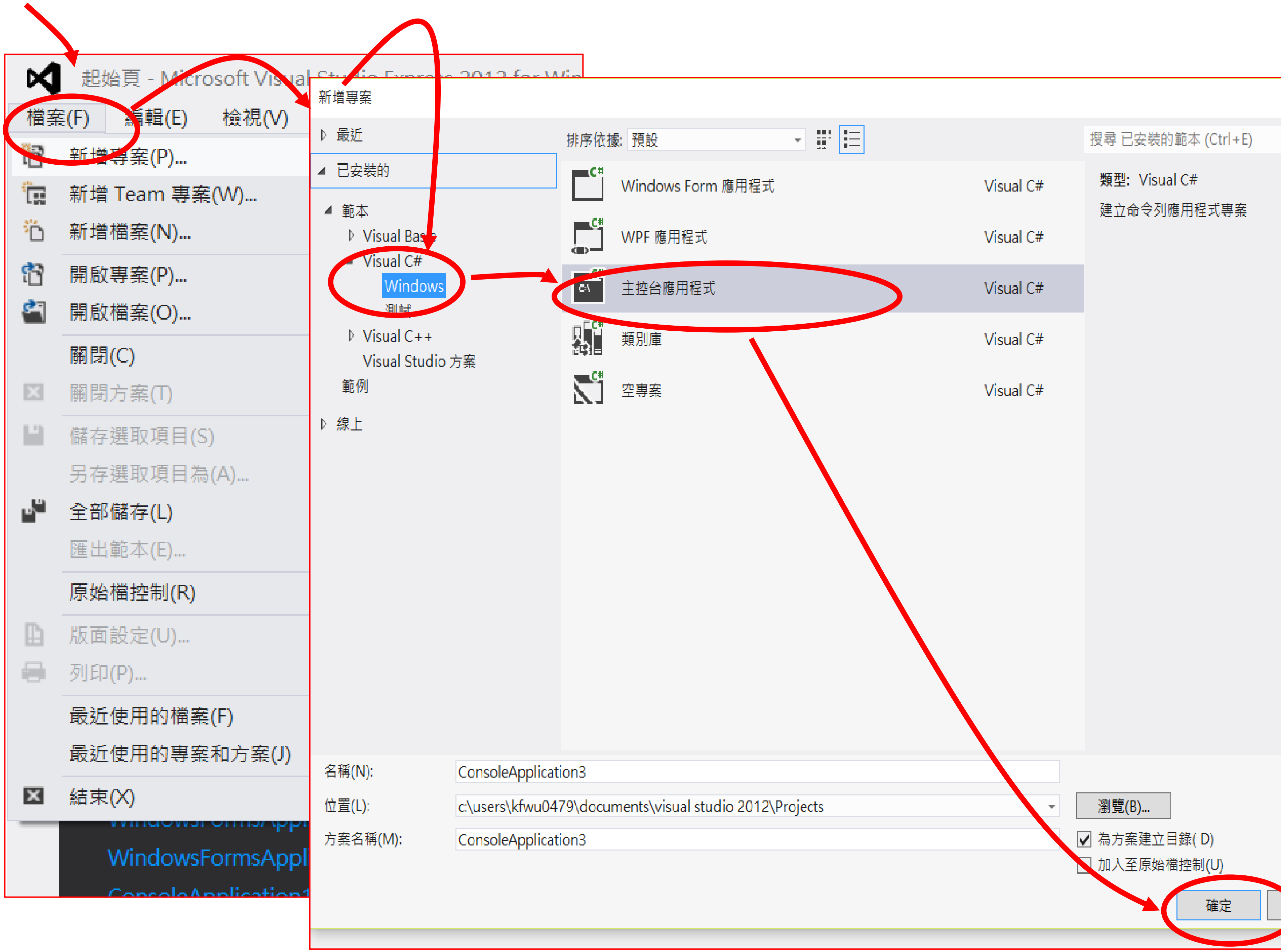
Method	Description	Example
Pow(x, y)	*x* raised to power *y* (xy)	pow(2.0, 7.0) is 128.0 pow(9.0, .5) is 3.0
Sin(x)	trigonometric sine of *x* (*x* is in radians)	sin(0.0) is 0.0
Sqrt(x)	square root of *x*	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
Tan(x)	trigonometric tangent of *x* (*x* is in radians)	tan(0.0) is 0.0

Math class methods.

Outline


最常使用
Google Chrome
Microsoft Word 2010
Microsoft PowerPoin...
VS Express for Desktop
小算盤
小畫家
檔案總管
設定
開啟/關閉
所有應用程式


起始頁 - Microsoft Visual Studio Express 2012 for Windows Desktop
檔案(F)
編輯(E)
檢視(V)
偵錯(D)
小組(M)
工具(T)
測試(S)
視窗(W)
說明(H)
附加...
工具箱
起始頁
Express 2012 for Windows Desktop
開始
新增專案...
開啟專案...
連接到 Team Foundation Server...
最近
PictureBoxTest
WindowsFormsApplication3
WindowsFormsApplication2
ConsoleApplication1
WindowsFormsApplication1
WindowsApplication2
開始使用
最新消息
歡迎
Windows Desktop Developm
Visual C# 的新功能
Visual Basic 的新功能
Visual C++ 的新功能
Windows Presentation Foundation (W
Windows Form 的新功能
Windows Desktop Applicatio
了解如何建置 Win32 / Visual C++ 應用
了解如何建置 WPF 應用程式
了解如何建置 Windows Form 應用程式
探索擴充功能、附加元件和範例
學習資源
Win32 的應用程式開發介面參考
WPF 應用程式的應用程式開發介面參考



類型: Visual C#

建立命令列應用程式專案

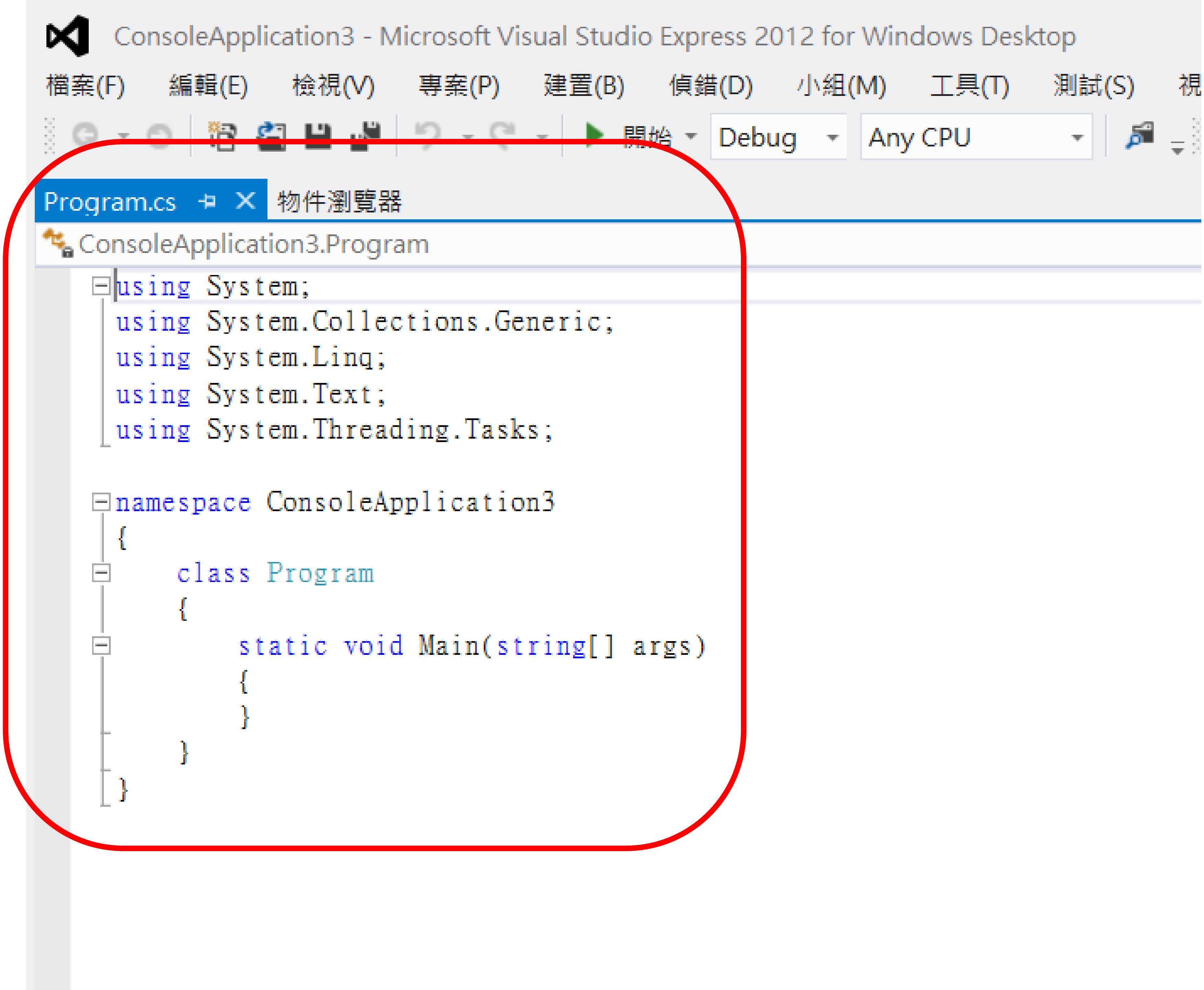
名稱(N): ConsoleApplication3

位置(L): c:\users\kfwu0479\documents\visual studio 2012\Projects

方案名稱(M): ConsoleApplication3

☑ 為方案建立目錄(D)

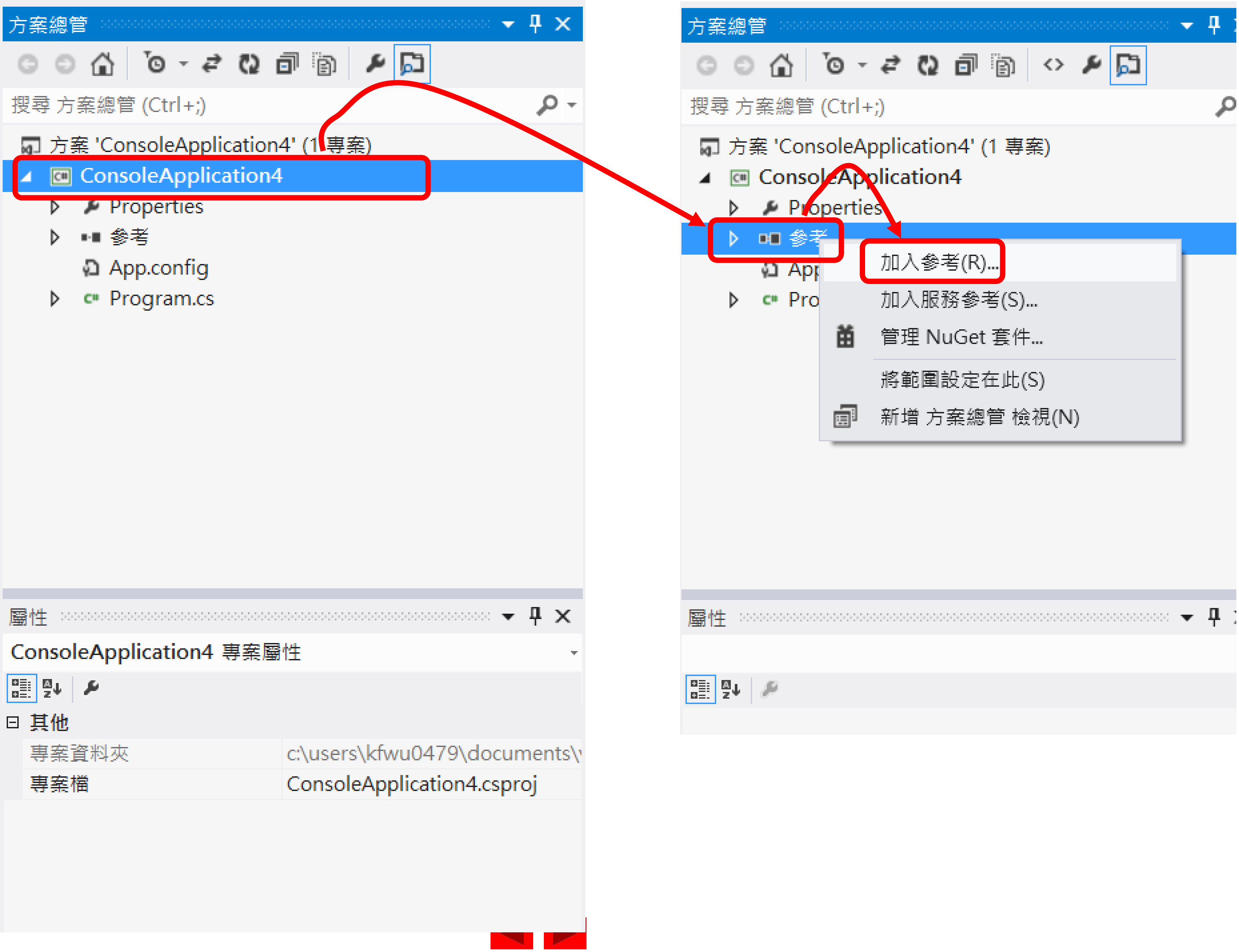
□ 加入至原始檔控制(U)



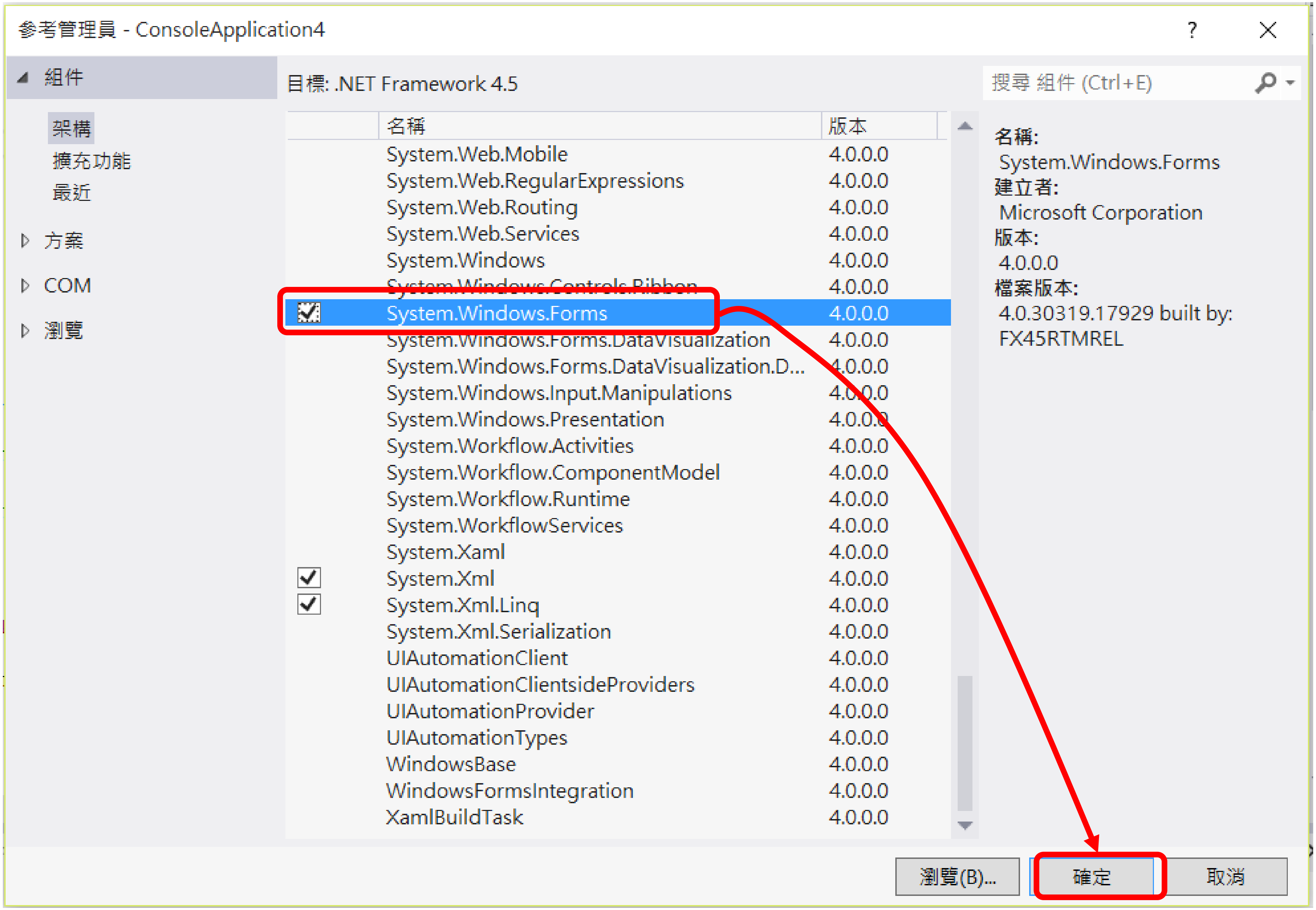
ConsoleApplication3 - Microsoft Visual Studio Express 2012 for Windows Desktop
檔案(F) 編輯(E) 檢視(V) 專案(P) 建置(B) 偵錯(D) 小組(M) 工具(T) 測試(S) 視
開始
Debug
Any CPU
Program.cs
物件瀏覽器
ConsoleApplication3.Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```



專案資料夾	c:\users\kfwu0479\documents\
專案檔	ConsoleApplication4.csproj



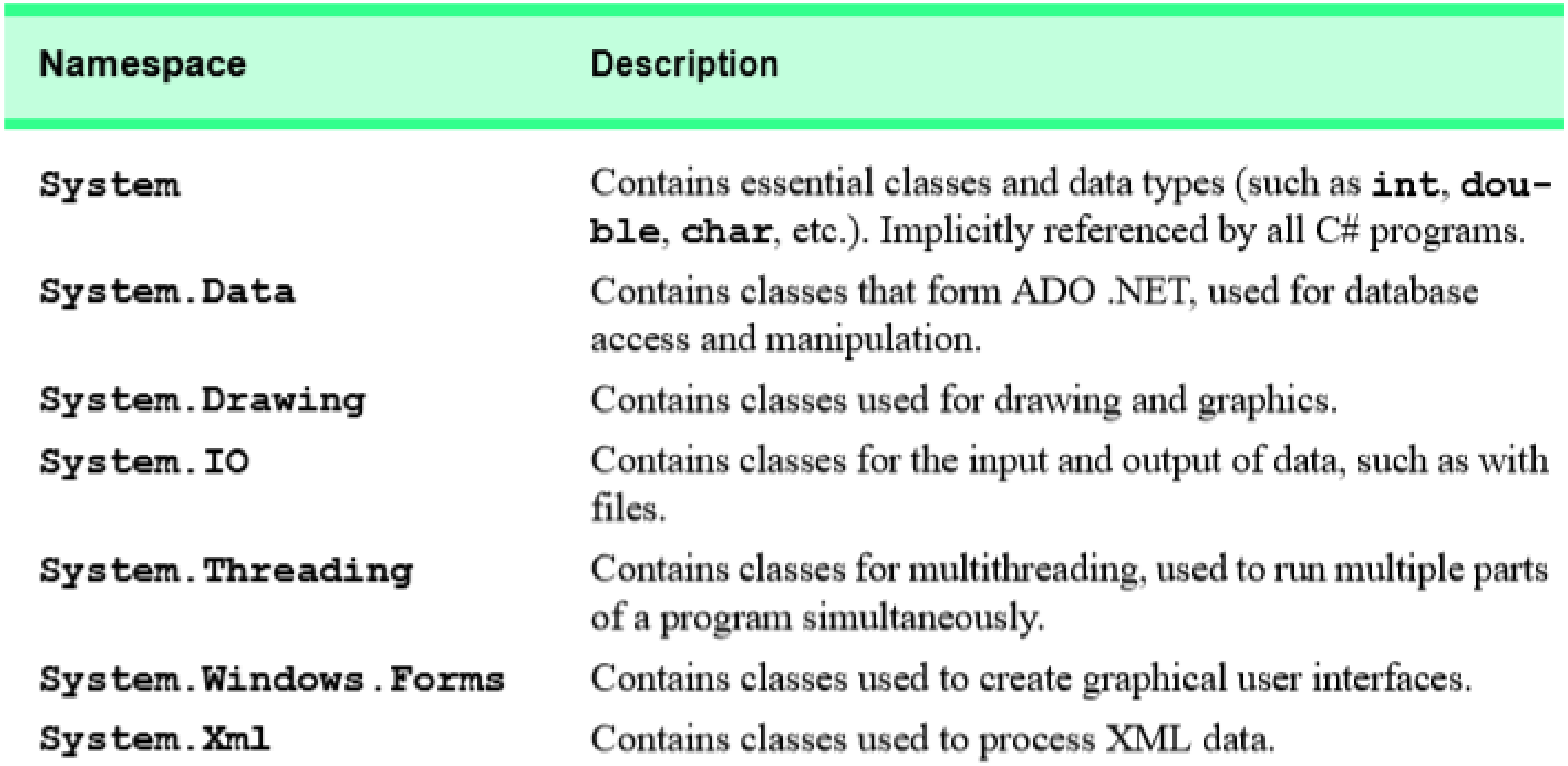
參考管理員 - ConsoleApplication4

目標: .NET Framework 4.5

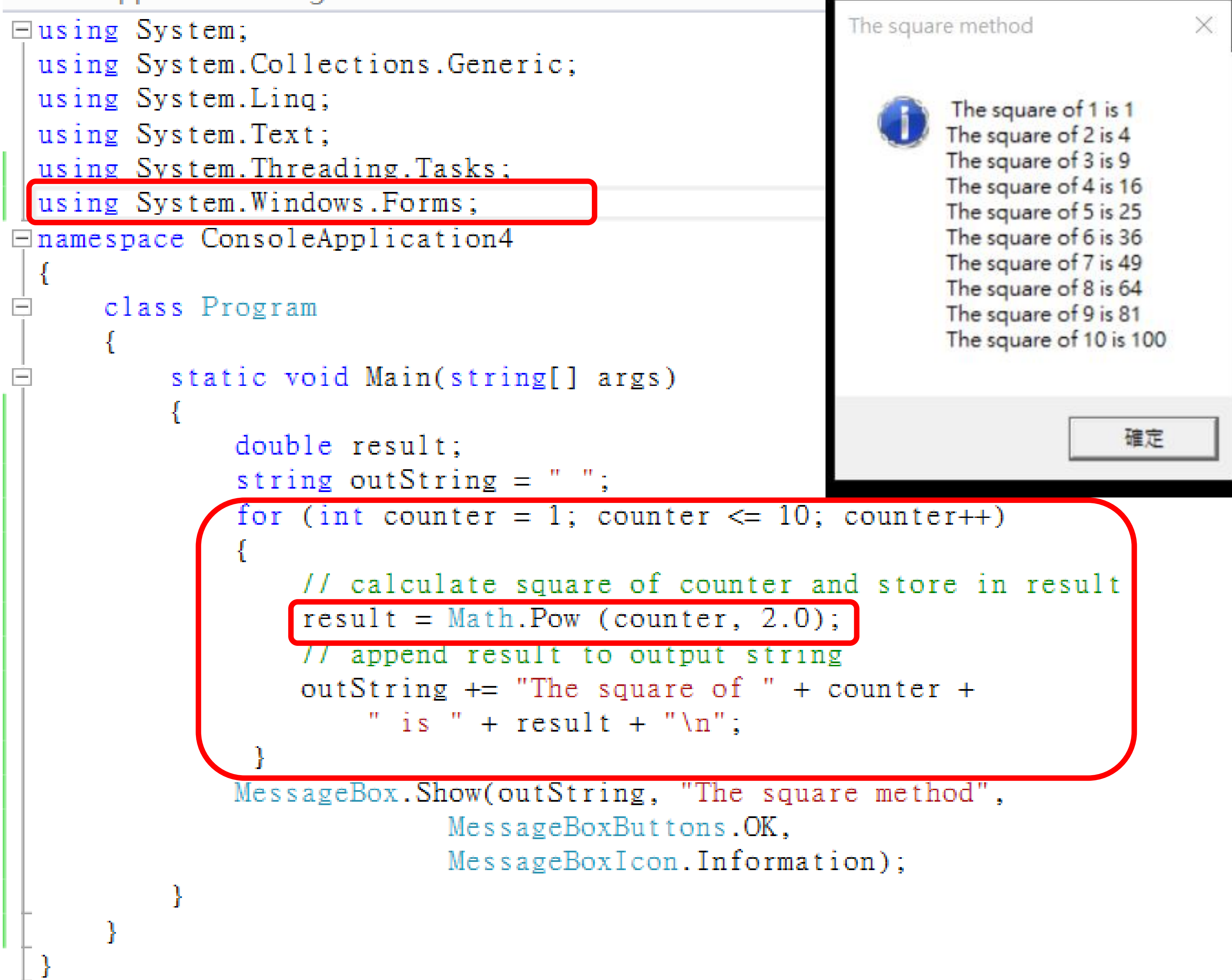
	名稱	版本
	System.Web.Mobile	4.0.0.0
	System.Web.RegularExpressions	4.0.0.0
	System.Web.Routing	4.0.0.0
	System.Web.Services	4.0.0.0
	System.Windows	4.0.0.0
	System.Windows.Controls.Ribbon	4.0.0.0
☑	System.Windows.Forms	4.0.0.0
	System.Windows.Forms.DataVisualization	4.0.0.0
	System.Windows.Forms.DataVisualization.D...	4.0.0.0
	System.Windows.Input.Manipulations	4.0.0.0
	System.Windows.Presentation	4.0.0.0
	System.Workflow.Activities	4.0.0.0
	System.Workflow.ComponentModel	4.0.0.0
	System.Workflow.Runtime	4.0.0.0
	System.WorkflowServices	4.0.0.0
	System.Xaml	4.0.0.0
☑	System.Xml	4.0.0.0
☑	System.Xml.Linq	4.0.0.0
	System.Xml.Serialization	4.0.0.0
	UIAutomationClient	4.0.0.0
	UIAutomationClientsideProviders	4.0.0.0
	UIAutomationProvider	4.0.0.0
	UIAutomationTypes	4.0.0.0
	WindowsBase	4.0.0.0
	WindowsFormsIntegration	4.0.0.0
	XamlBuildTask	4.0.0.0

名稱:
System.Windows.Forms
建立者:
Microsoft Corporation
版本:
4.0.0.0
檔案版本:
4.0.30319.17929 built by: FX45RTMREL

Example of C# system's name space



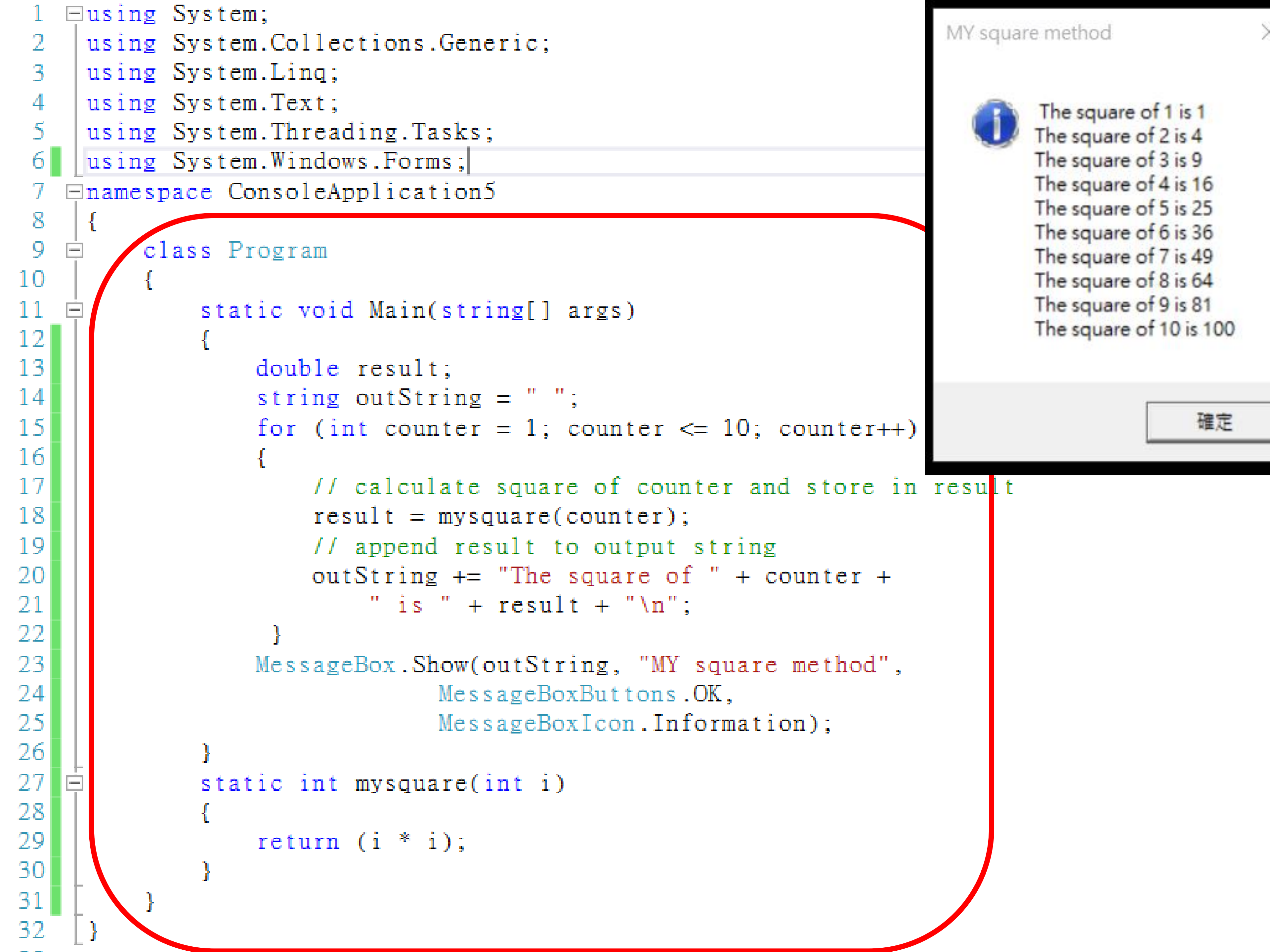
Namespace	Description
`System`	Contains essential classes and data types (such as `int`, `double`, `char`, etc.). Implicitly referenced by all C# programs.
`System.Data`	Contains classes that form ADO .NET, used for database access and manipulation.
`System.Drawing`	Contains classes used for drawing and graphics.
`System.IO`	Contains classes for the input and output of data, such as with files.
`System.Threading`	Contains classes for multithreading, used to run multiple parts of a program simultaneously.
`System.Windows.Forms`	Contains classes used to create graphical user interfaces.
`System.Xml`	Contains classes used to process XML data.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            double result;
            string outString = " ";
            for (int counter = 1; counter <= 10; counter++)
            {
                // calculate square of counter and store in result
                result = Math.Pow (counter, 2.0);
                // append result to output string
                outString += "The square of " + counter +
                    " is " + result + "\n";
             }
            MessageBox.Show(outString, "The square method",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Information);
        }
    }
}
```

The square method

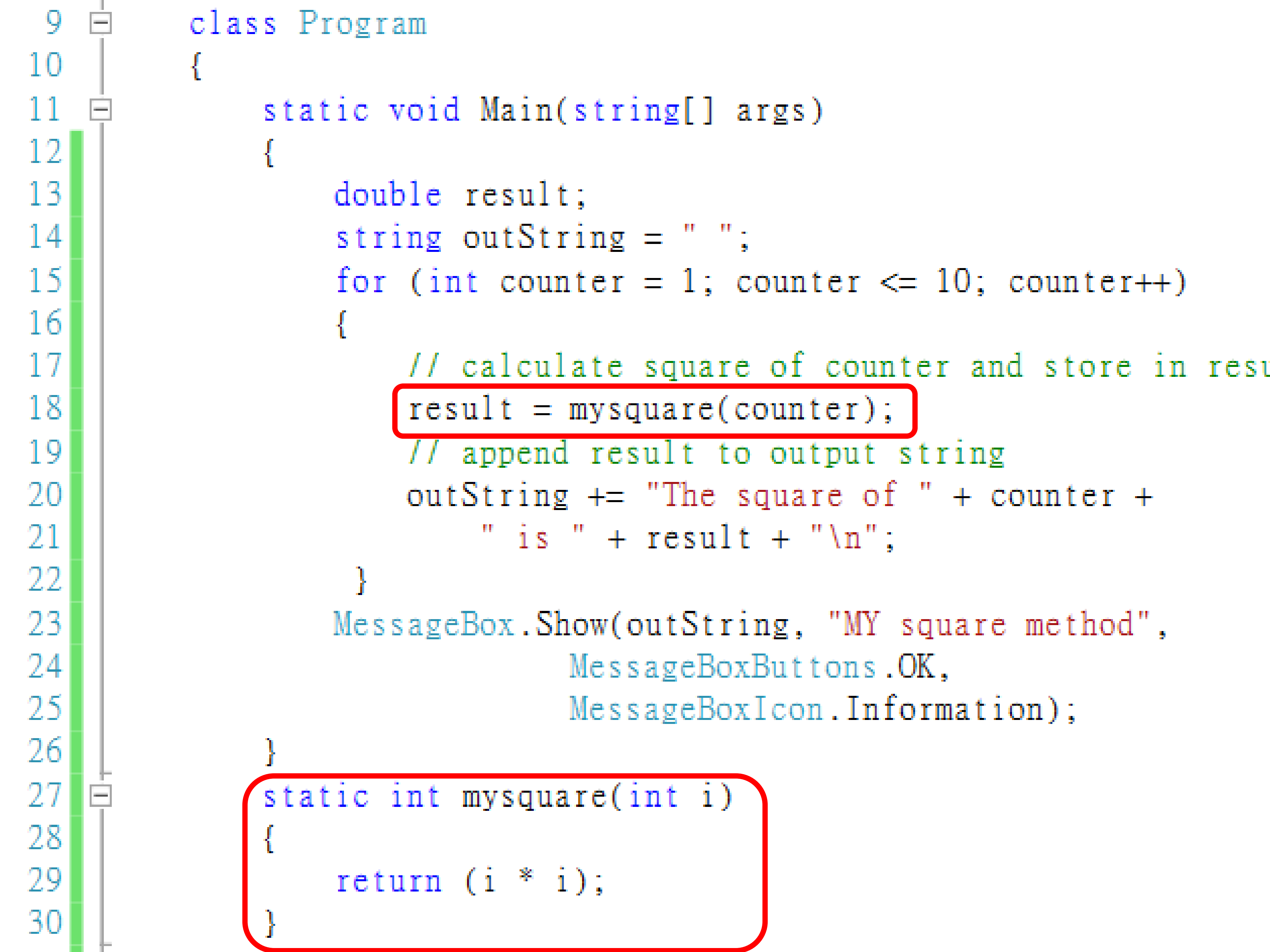
```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            double result;
            string outString = " ";
            for (int counter = 1; counter <= 10; counter++)
            {
                // calculate square of counter and store in result
                result = mysquare(counter);
                // append result to output string
                outString += "The square of " + counter +
                    " is " + result + "\n";
             }
            MessageBox.Show(outString, "MY square method",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
        }
        static int mysquare(int i)
        {
            return (i * i);
        }
    }
}
```

MY square method

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```



```
    class Program
    {
        static void Main(string[] args)
        {
            double result;
            string outString = " ";
            for (int counter = 1; counter <= 10; counter++)
            {
                // calculate square of counter and store in resu
                result = mysquare(counter);
                // append result to output string
                outString += "The square of " + counter +
                    " is " + result + "\n";
             }
            MessageBox.Show(outString, "MY square method",
                         MessageBoxButtons.OK,
                         MessageBoxIcon.Information);
        }
        static int mysquare(int i)
        {
            return (i * i);
        }
```

Static Methods

- static method can be seen as a universal-wide method that you can call it at will
  - Declaration syntax:

    ex: static int square(float w) { ……}
  - Before you call it, you must import (using) the package containing it
    - you do not need to create any object of the class (discussed later)
  - When you call it, you must write the class name before the method name
    - Like **Math.Abs(-4), which will return 4**
  - **you can think static method always exist that you can use it**
- **Given that static method always be there, why static method needs to be contained in a class?**
  - **For easily maintained and classified**

Discussion of Methods (1/3)

- Legend of method
  - In the past, a method can be divided into procedure and function
    - A procedure does not return value to the caller
      - ex. In pascal, Ada, delphi, declaration syntax for procedure is

```
procedure sorting(integer  kk[])
   begin
    ….      end;
```

    - A function should return a value to the caller
      - ex. In pascal, Ada, delphi, declaration syntax for function is

```
function average(integer  kk[]): real
     begin
          ……__ __end;
```

Discussion of Methods 2/3

- Legend of method
  - In C-like language, it does not distinguish function and procedure in declaration; i.e., it uses the similar syntax in declaration
    - Ex: function declaration

      ```
      long max (long n1, n2) {
           .... }
      ```

    - Ex: procedure declaration

      ```
      void sorting(int mm[]) { .....   }
      ```

  - If a method is used as a procedure, its call is:

    ```
    sorting(mm);  // where mm is an array
    ```

  - If a method is used as a function, its call is:

    ```
    results = average(mm); // where mm is an array
    ```

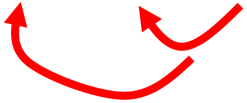
Discussion of Methods 3/3

- Note: In C-like language, each statement should return a value.
  - Eq 1: If (a = 5)

    ```
          ......;
    ```

    // a = 5 will return 5 (which is seen as true)

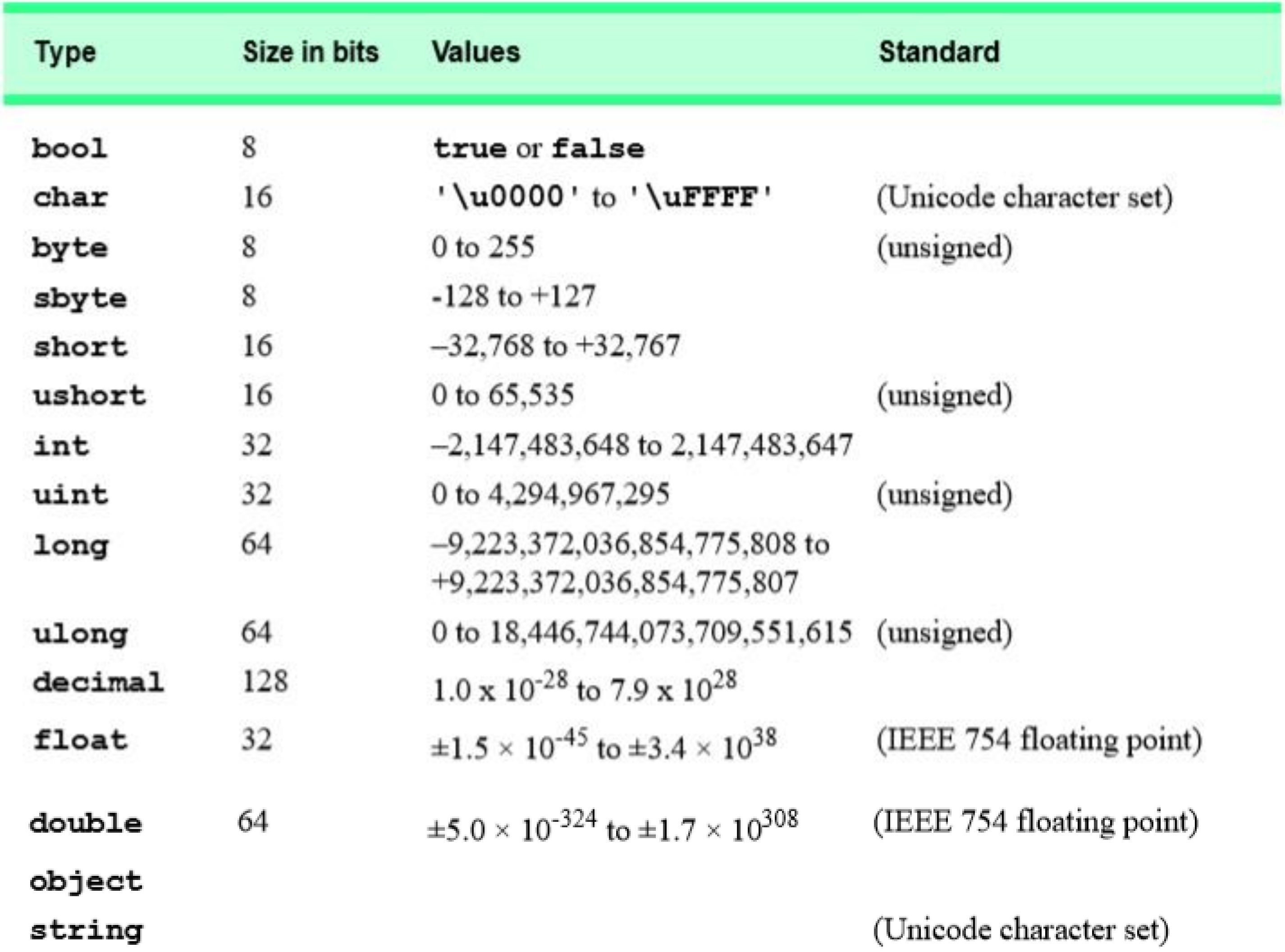
    // in C, C#, 0 is false; non-zero is true
  - Eq 2: b = a = 5;



// a = 5 will return 5, and then 5 is assigned to b.

◀ ▶

Variable's property: memory size



Type	Size in bits	Values	Standard
`bool`	8	`true` or `false`	
`char`	16	`'\u0000'` to `'\uFFFF'`	(Unicode character set)
`byte`	8	0 to 255	(unsigned)
`sbyte`	8	-128 to +127	
`short`	16	–32,768 to +32,767	
`ushort`	16	0 to 65,535	(unsigned)
`int`	32	–2,147,483,648 to 2,147,483,647	
`uint`	32	0 to 4,294,967,295	(unsigned)
`long`	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	
`ulong`	64	0 to 18,446,744,073,709,551,615	(unsigned)
`decimal`	128	$1.0 \text{ x } 10^{-28}$ to $7.9 \text{ x } 10^{28}$	
`float`	32	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	(IEEE 754 floating point)
`double`	64	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	(IEEE 754 floating point)
`object`			
`string`			(Unicode character set)

Variable's property

- A variable has two properties:
  - Scope (location concept)
    - means what a variable is referred when the program refers to
      - Ex. suppose our class has a student called 柯P. When I called 柯P, which one I refer to?
      - If I called 柯P in class, I mean our classmate; if I called it outside the class, I mean Taipei's mayor
  - Lifetime (life concept)
    - Means how long the variable will exist
      - Ex. real estate is scarce. Its ownership can be permanent or temporary.
        - Compare Taiwan's policy with China in real estate

Local Variable (1/5)

- Local variable
  - In scope concept:
    - Shield from outside intrusion
    - Does not need to remember the legend of the a variable happened outside the scope
    - May be suitable for a team work
  - In life concept:
    - Once outside of the scope it is defined, the variable is deallocated (its memory is recycled)
    - Can reduce the need of the memory
  - Advantage
    - Information hiding
    - Reduce the tracing efforts for a variable
  - Disadvantage:
    - Misused by apprentice

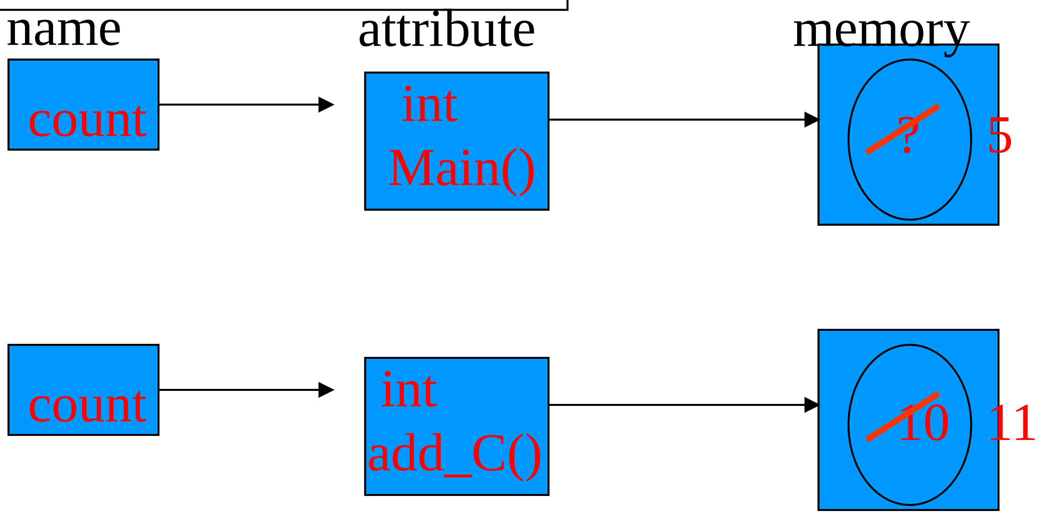
Local Variable (2/5)

- Syntax of declaration
  - Declare within a block (eq. for loop) or in a method
  - Sometimes we call local variable as auto variable (in life concept)
- Life concept (不在乎天長地久,只在乎曾經擁有)
  - Its life begins from the execution of the block (or method) and last until the end of the block (or method) the variable is declared
  - OS automatically allocates a memory to it when the execution enters the block the variable is declared,
  - OS automatically deallocates the memory of the variable when the execution exits the method (block) the variable is declared
- Scope concept (家醜不外揚)
  - It can only be seen and used in the declaring block or method

Local Variable (3/5)

```
Class test2{
        static void Main () {
             int count;
             count = 5;
             add_C();
             add_C();
             add_C();
        } }
```

```
        private add_C() {
             int count = 10;
             count++;
        }
```



name

attribute

memory

count

int
Main()

~~?~~ 5

count

int
add_C()

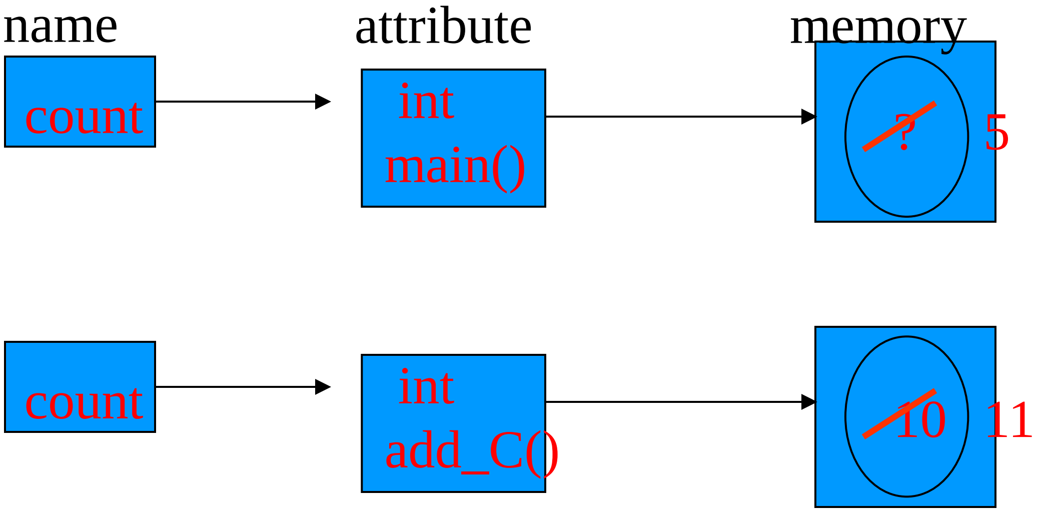
~~10~~ 11

◀ ▶

Local Variable (4/5)

```
Class test2{
    static void Main () {
        int count;
        count = 5;
        add_C();
        add_C();
        add_C();
    }
```

```
private add_C() {
    int count = 10;
    count++;
}
```



name

attribute

memory

count

int
main()

~~?~~ 5

count

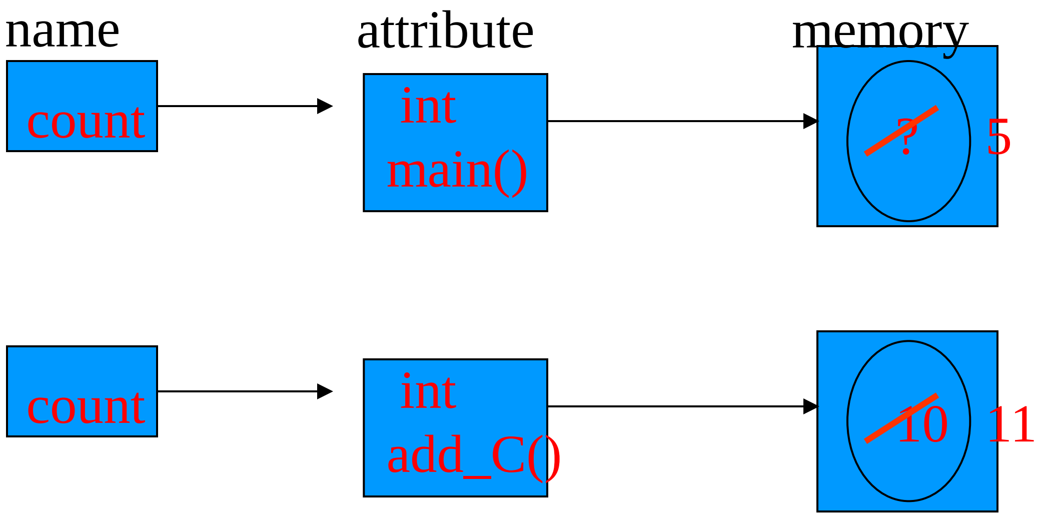
int
add_C()

~~10~~ 11

Local Variable (5/5)

```
Class test2{
      static void Main () {
            int count;
            count = 5;
            add_C();
            add_C();
            add_C();
      }
```

```
      private add_C() {
            int count = 10;
             count++;
      }
```



Global Variable (or class-wide variable)

- Syntax of declaration
  - Declare inside a class but outside its method
  - In C#, Java, we can call global variable as class-wide variable (scope concept)
- Life concept (海枯石爛)
  - Its life begins from the declaration and last until the end of the program
  - OS automatically allocates a memory to it when the execution instantiate (or new) an object of the class,
  - OS automatically deallocates the memory of the variable when the end of the object
- Scope concept (你的就是我的,我的就是你的)
  - More than one method in the same class can see and use the variable

◀ ▶

Global variable

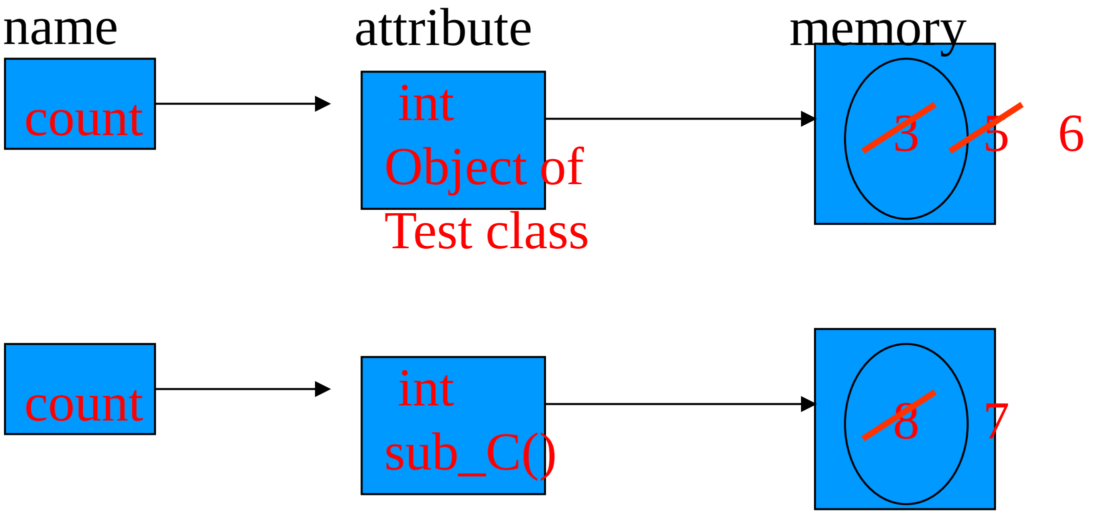
- Global variable
  - In scope concept:
    - Every method in its class refers to the same variable
    - reduce the passing of parameters among the methods in the class
  - In life concept:
    - Can remember the change of a variable
  - Advantage
    - Reduce the passing of parameters
  - Disadvantage:
    - need to remember the legend of the a variable

Global variable

```
Class test() {
    int count =3 ;
    static void Main () {
        count = 5;
        add_c();
        sub_c();
}
```

```
public void add_c() {
    count++; }
```

```
private void sub_c() {
    int count = 8;
    count--;
  }
}
```



Hole in scope

- Global variable cannot be seen if a local variable with the same name exists
  - In the last example, variable count in sub_c( ) refers to its local variable, not global variable
  - In such condition, if wanting to change the value of global variable, full name should be used,

    eq: test.count--;
- Alternative variable (discussed later)
  - Static local variable
  - Static global variable
  - Static variable
  - Reference variable

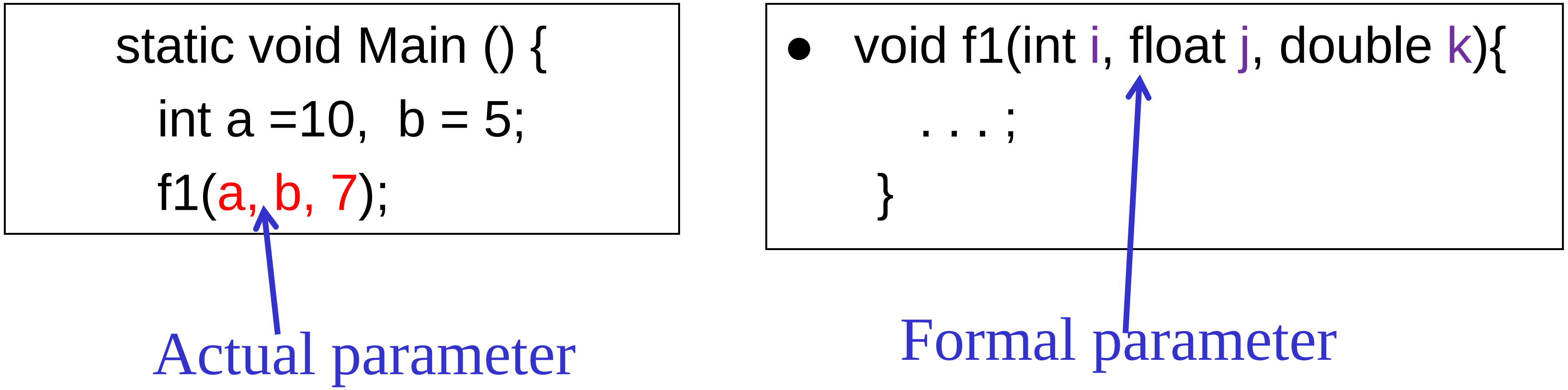
◀ ▶

Discussion of parameter in methods

- Parameters
  - Used to communicate information between methods
  - **Can be seen as local auto variable (since it is defined in the method)**
    - Scope: only can be seen and used within this method
    - Life time: begin from the execution of the method, over until the end of the method
- Two types of parameters:
  - formal parameter
    - parameter defined in the method declaration
  - Actual parameter
    - the value (variable) used in the calling statement

◀ ▶

Discussion of parameter



```
static void Main () {
    int a =10,  b = 5;
    f1(a, b, 7);
```

Actual parameter

```
•  void f1(int i, float j, double k){
       . . . ;
   }
```

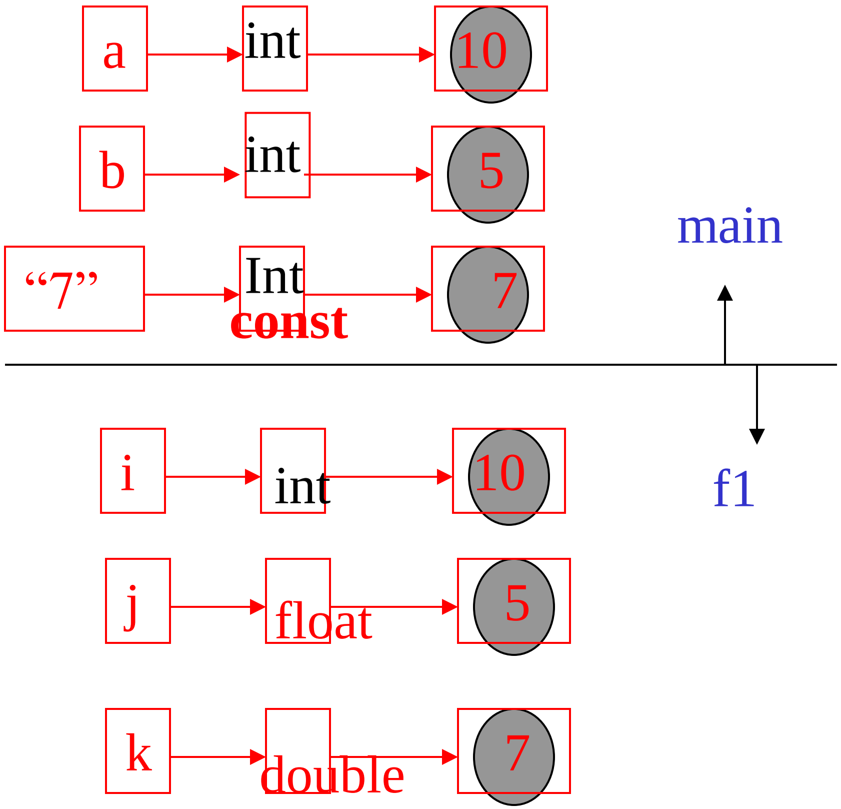
Formal parameter

- Can the method change the value of the actual parameter?
  - Depends

◀ ▶

Call-by-value (I)

- Goal (銀貨兩訖)
  - Method does not change the value of actual parameter, eq:
- Memory status



```
static void Main () {
   int a =10, b = 5;
   f1(a, b, 7);
```

```
void f1(int i, float j, double k){
       . . . ;
    }
```

◀ ▶

Call-by-value (II)

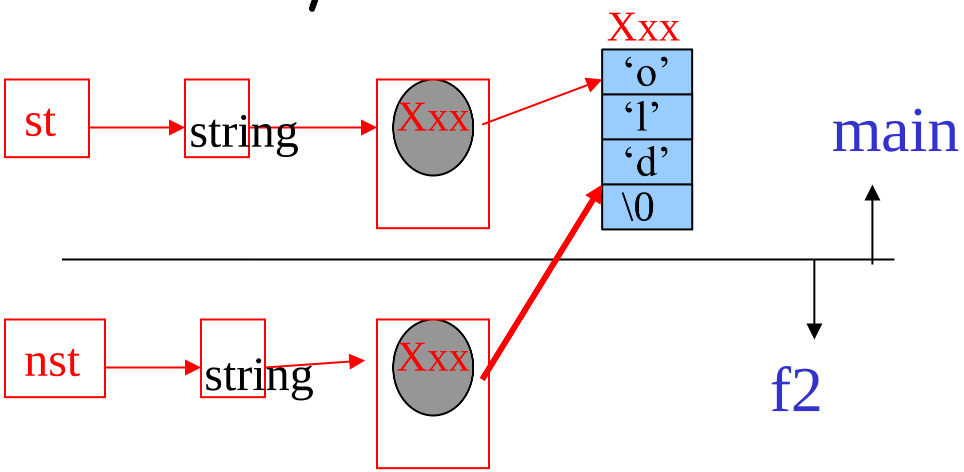
- Advantage
  - Caller needs not to know how the ‘called’ works on the parameter,
  - the results in the called will not change the caller’s status
- Disadvantage
  - ‘Called’ needs extra space to store the formal parameter

- In C#, the primitive data (like int, float, double, Boolean) is default to use call-by-value, except the prefix “ref” is used.

Call by reference (I)

- Goal (你泥中有我，我泥中有你, or 概括承受)
  - method use the same data memory as the called
  - Method can directly change the value of actual parameter

- Memory status

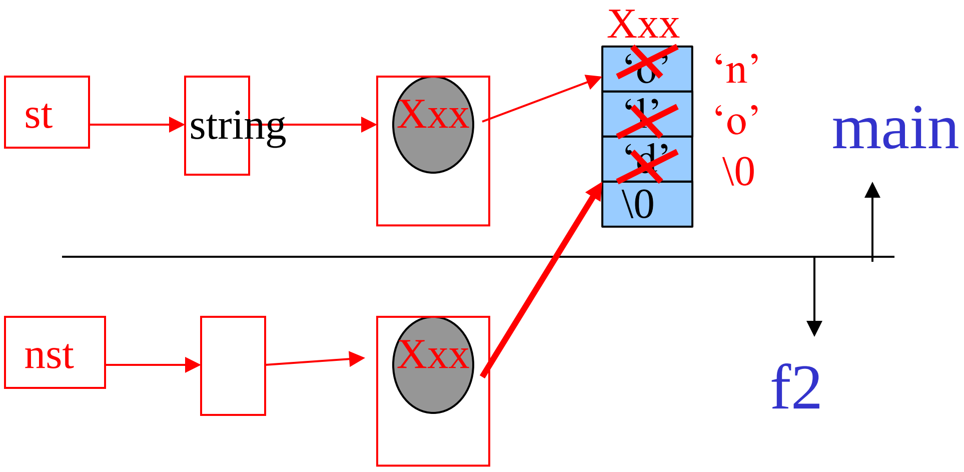


```
        static void main () {
           string st =“old”;
           f2(st);
```

```
void f2(string nst){
          . . . ;
          nst = “no”;
      }
```

Call by reference (II)

- Memory status



```
static void main () {
    string st =“old”;
    f2(st);
```

```
void f2(string nst){
        . . . ;
        nst = “no”;
    }
```

Call by reference (III)

- Advantage
  - No need of extra space (for data memory)
    - this property is useful for array, large reference data variable
- Disadvantage
  - Caller needs to know how the ‘called’ works on the parameter
  - Note: Constant variable cannot be used as such a parameter (since the called may change its value)

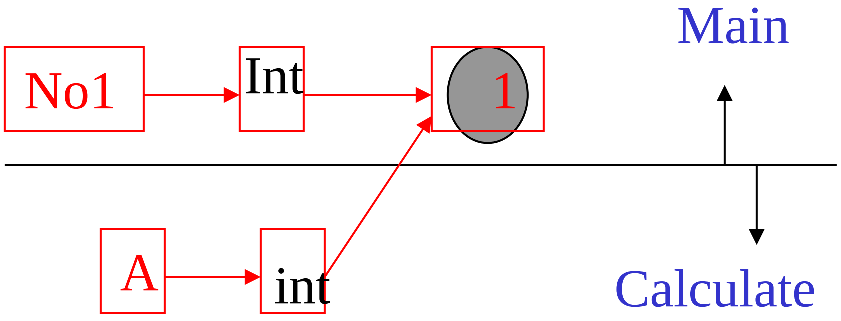
- In *C#*, the reference data (like string, array, object) is default to use call-by-reference

◀ ▶

Variation of call-by-reference

- ref
  - C# provides the indicator (i.e., ref) to explicitly use call-by-reference,
    - which can change the default call-by-value into call-by-reference
  - Eq:

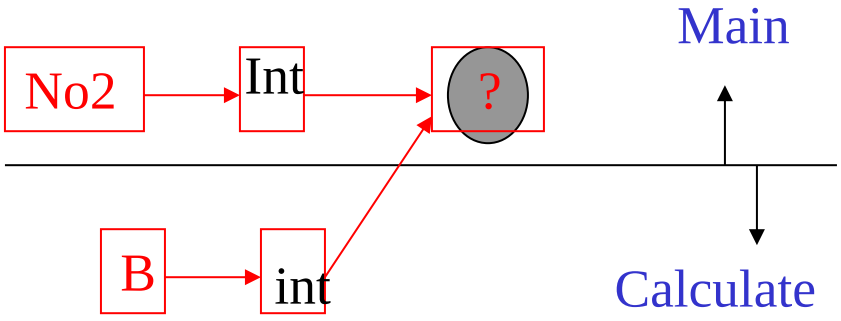
```
class ExampleOutAndRef {
      static void Calculate(ref int A) {
            A = A + 1;   }
      static void Main() {
            int No1 = 1;
            Calculate(ref No1);
            Console.WriteLine("No1:" + No1);
       }
 }
```



Variation of call-by-reference

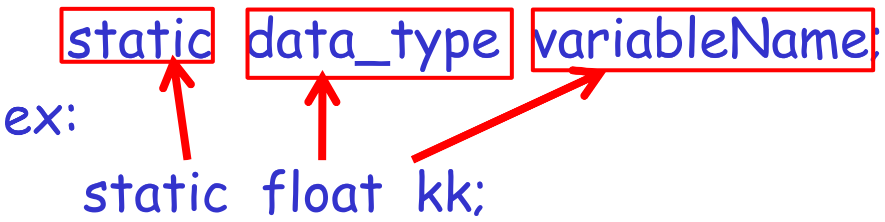
- out
  - C# provides another indicator (i.e., out) to explicitly use call-by-reference,
    - Out can allow the parameter not be initialized when passed
  - Eq:

```
class ExampleOutAndRef {
    static void Calculate(ref int B) {
        B = 20;   }
    static void Main() {
        int No2;
        Calculate(ref No2);
        Console.WriteLine("No2:" + No2);
    }
}
```



Variables (parameters) within Methods

- Issues in variables (parameters) defined within Methods:
  - 1st dimension: primitive or reference variable
    - Having discussed in chaps 4 & 5
  - 2nd dimension: global or local variable:
    - Global variable (class-wide variable)
      - Declared outside the method but within the class
    - Local variables (method-wide variable)
      - Declared within the method definition
  - 3rd dimension: static, or auto (or non-static)
    - static variable
      - Syntax:



static data_type variableName;

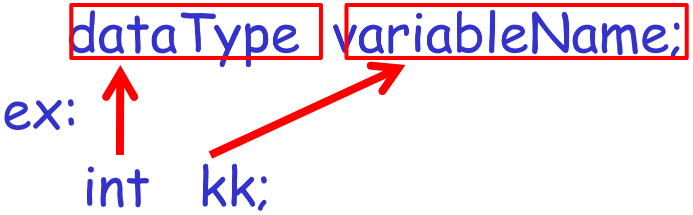
ex:

static float kk;

- Cf: Variable definition without static is auto variable

Variables (parameters) within Methods

- Memory for static variable
  - be allocated after compilation
  - not to be deallocated during and after the execution
- Non-static variable (auto variable)
  - Syntax:



dataType variableName;

ex:

int kk;

  - Memory for auto variable
    - not to be allocated until enter (i.e., execute) the block the variable is defined
    - to be deallocated just when the exit the block (or method) the variable is defined

- What if the three dimensions (i.e., global (local), primitive (reference), and static (auto)) mingle together?
  - Follows the orthogonal property

orthogonal property

- orthogonal property
  - Ideal property that many system (such as languages, medicine) hope to own. The following is not orthogonal property
    - 咱要騎自行車去銀行領錢;那本書很便宜，如果方便，送我一本
- C-like language strictly pursuits orthogonal property,
  - no side effect occurs when two properties mingling together
    - You can use “if statement” in “while statement”, and the “if statement” can have “for statement” in it, etc
- SQL statement not guarantee orthogonal property

Variables (parameters) within Methods

- static local variable
  - Declared within the method and with static before it
    - Ex:

```
float fun1( ) {
        static int  j;

        …….
```

  - memory of the variable exists when the program starts to execute and will not be deallocated (reclaimed) during the execution
    - Static local variable is used to accumulate the results when a method is called for many times
  - A statement cannot use the local static unless the statement is executed in the block (or method) the variable is defined

◀ ▶

Variables (parameters) within Methods

- Static global variable
  - Declared outside the method definition, but within a class, and with static before it
    - Ex:

```
class test( ) {
    static int i, k;
   float fun1( ) {
       static int  j; .....}  }
```

  - memory of the variable exists when the program starts to execute and will not be deallocated (revoked) during the execution
    - Static global variable is used when no object of the class is instantiated (discussed later)
  - A statement can use it anywhere no matter whether the statement is in the block (method) the variable is defined

    Ex.   Math.Pie is to get the static global variable Pie

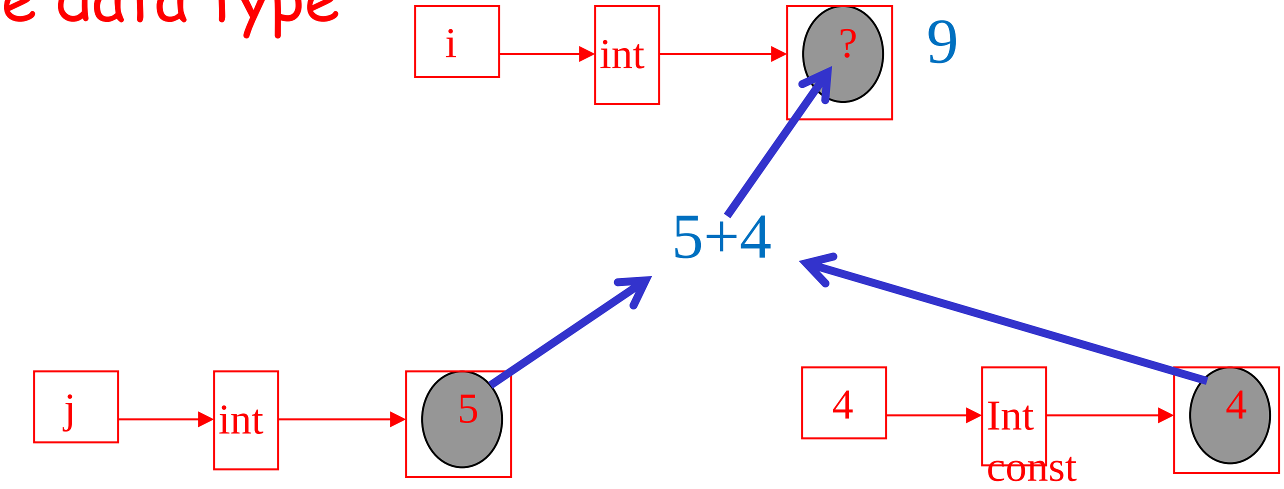
Lvalue and Rvalue of a variable

- R-value:
  - Variable appear in the right of assignment
- L-value:
  - Variable appears in the left of assignment

Ex: int i, j = 5;

i = j +4; //i is L-value; j and 4 are R-values

- R-values get their contents in the third box and put the results to L-vaule's third box
  - The rule is the same for the primitive data type and reference data type



Argument Promotion

- Coercion of arguments
  - Forcing parameters to appropriate type while passing to method
  - Implicit Conversion: no indicator
  - Explicit Conversion: with indicator
- Promotion rules
  - Implicitly convert types without data loss
- Demotion must be specified (called 'cast')
  - The demotion data type should be added before the variable (or return value)

◀ ▶

Type	Can be Converted to Type(s)
`bool`	`object`
`byte`	`decimal`, `double`, `float`, `int`, `uint`, `long`, `ulong`, `object`, `short` or `ushort`
`sbyte`	`decimal`, `double`, `float`, `int`, `long`, `object` or `short`
`char`	`decimal`, `double`, `float`, `int`, `uint`, `long`, `ulong`, `object` or `ushort`
`decimal`	`object`
`double`	`object`
`float`	`double` or `object`
`int`	`decimal`, `double`, `float`, `long` or `object`
`uint`	`decimal`, `double`, `float`, `long`, `ulong`, or `object`
`long`	`decimal`, `double`, `float` or `object`
`ulong`	`decimal`, `double`, `float` or `object`
`short`	`decimal`, `double`, `float`, `int`, `long` or `object`
`ushort`	`decimal`, `double`, `float`, `int`, `uint`, `long`, `ulong` or `object`

Fig. 6.5 Allowed implicit conversions.

Recursion

- Recursive method
  - A method calls itself directly, or indirectly through another method
  - Method should
    - *reduce the big problem to smaller problem(s)*
    - solve the problem for a *base case*
  - That is, the method should divides problem into
    - Base case
    - Derived case but based on the previous case

◀ ▶

Recursion (II)

- Example: factorial:

  5! = 5 * 4 * 3 * 2 * 1

  Notice that

  5! = 5 * 4!

  4! = 4 * 3! ...

  - We can compute factorials recursively
  - We define the recursive function (the base case is (1! = 1)) then
    - **F(1) = 1;**
    - **F(n) = n * F(n-1);**

```
long factorial(long n){
   if (n==1)  //base case
     return 1;
   else return n*factorial(n-1);
}
```

Example Using Recursion: The Fibonacci Series

- Fibonacci series: 0, 1, 1, 2, 3, 5, 8...
  - Each number is the sum of the previous two, i.e.,

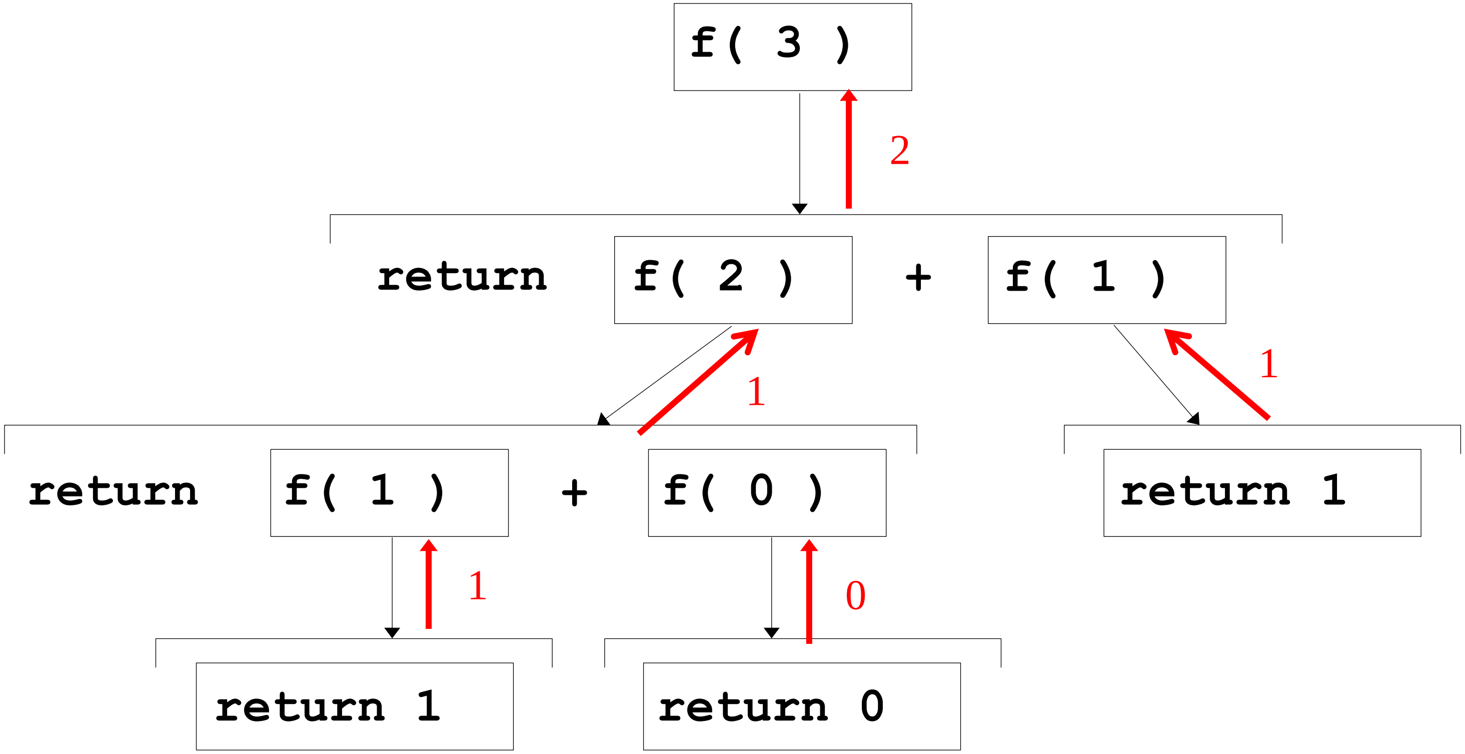
  fib(n) = fib(n-1) + fib(n-2)

  -- recursive formula with the base case is fib(0) =0, and fib(1) = 1;

```
long fibonacci(long n)
{
   if (n==0 || n==1)  //base case
        return n;
   else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

◀ ▶

Example Using Recursion: The Fibonacci Series (II)



```
long fibonacci(long n)
{
   if (n==0 || n==1)  //base case
        return n;
   else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

Recursion vs. Iteration (cont.)

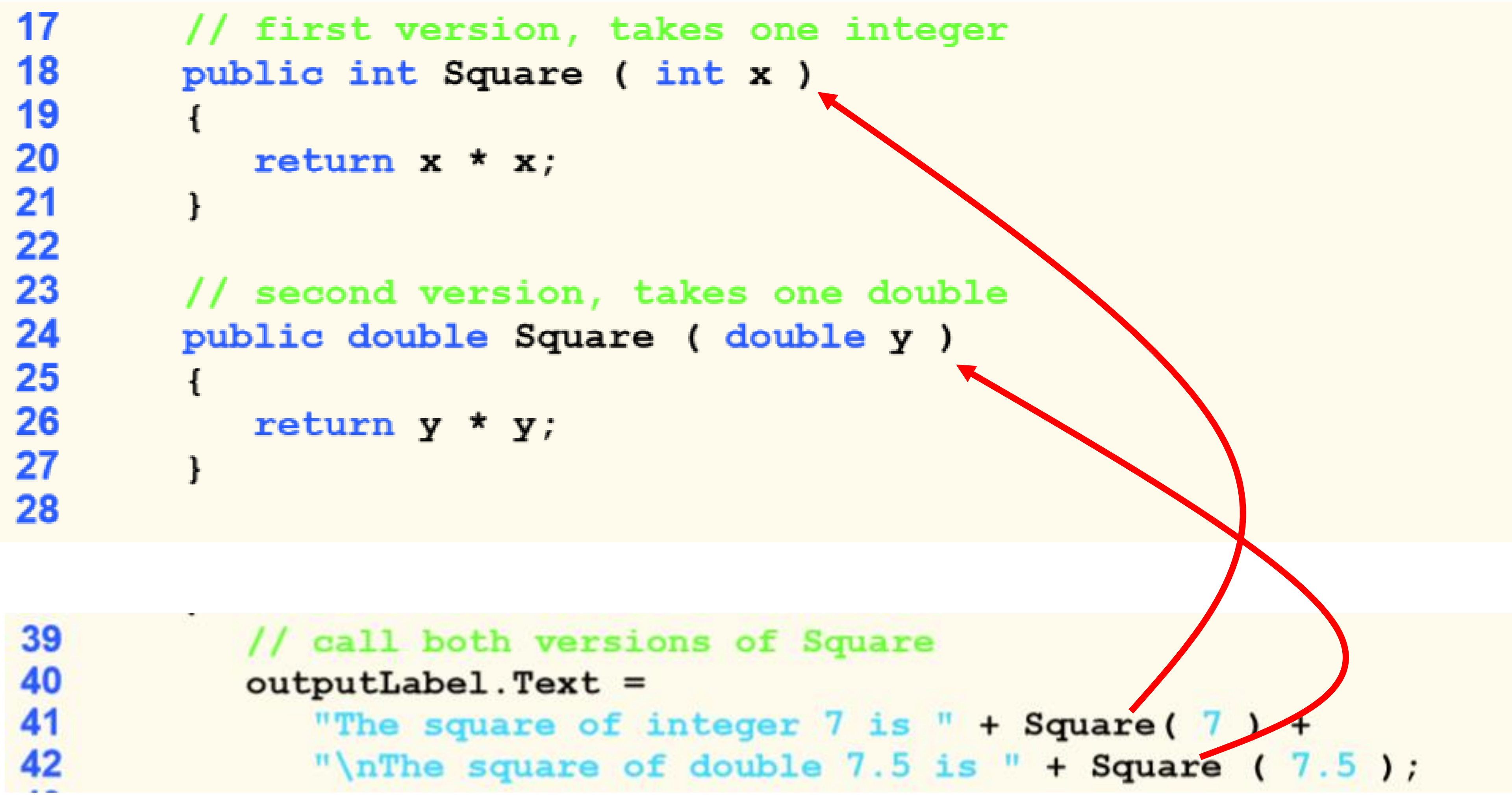
- Recursion
  - More overhead (means push and pop stacks) than iteration
  - More memory usage than iteration
  - Can also be solved iteratively
  - But often can be implemented with only a few lines of recursive code

Recursion vs. Iteration

- Iteration
  - Uses repetition structures (for, while or do…while)
  - Repetition through explicitly use of repetition structure
  - Terminates when loop-continuation condition fails
  - Controls repetition by using a counter
  - No stack used (discussed in data structure)
- Recursion
  - Uses selection structures (if, if…else or switch)
  - Repetition through repeated method calls
  - Terminates when base case is satisfied
  - Controls repetition by dividing problem into simpler one
  - Need stack (discussed in structure)

Method Overloading

- Methods in a class with the same name
  - We can save the trouble to think of different names for processing the similar jobs
  - Usually perform the same task
    - On different data types
  - But we need different arguments (types and number) when we define the methods with the same name
    - Variables passed must be different
      - Either in type received or order sent



```
      // first version, takes one integer
      public int Square ( int x )
      {
         return x * x;
      }

      // second version, takes one double
      public double Square ( double y )
      {
         return y * y;
      }

```

```
         // call both versions of Square
         outputLabel.Text =
            "The square of integer 7 is " + Square( 7 ) +
            "\nThe square of double 7.5 is " + Square ( 7.5 );
```