

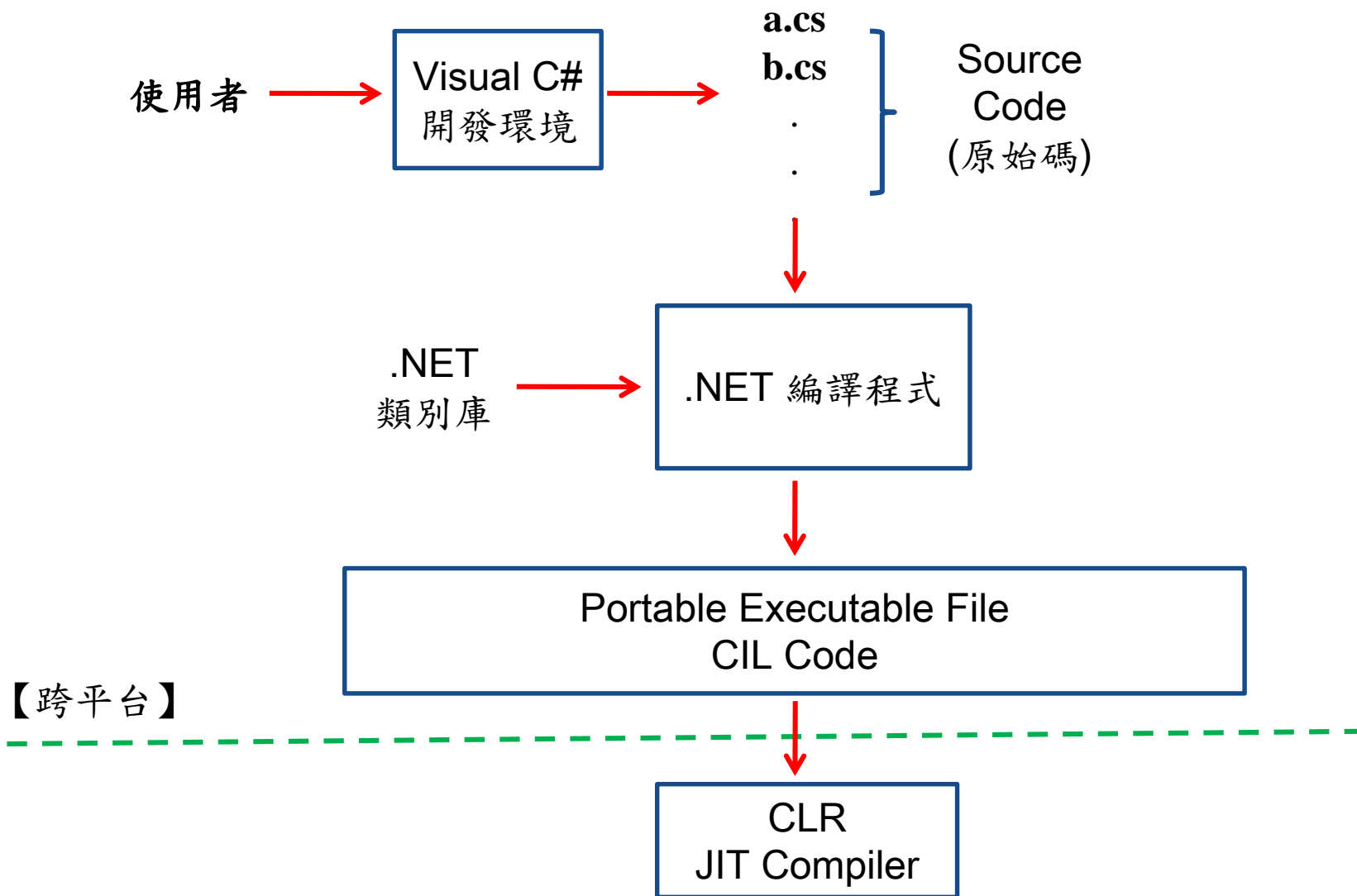
.NET Framework

- .NET Framework是**微軟**的軟體開發平台(軟體框架)，最早的 .NET Framework 1.0 於2002年發行
 - 支援多種程式語言開發，包括**Visual C#**、**Visual C++**、**Visual Basic**、**Python**等
 - .NET各種程式語言的原始碼會被翻譯成**CIL**碼(Common Intermediate Language，類似Java的bytecode)，再由**虛擬機器 CLR**(Common Language Runtime，類似Java的JVM)進行即時編譯(JIT)，可以在各種支援.NET的不同平台環境上執行
 - 2014年11月，微軟宣布完全開放.NET Framework的原始碼，並提供給Linux和Mac OS使用

Visual C#

- Visual C#是微軟推出支援.NET Framework的物件導向程式語言，可進行視窗介面的軟體開發
 - C#的發音為C sharp，取自音樂的C#(C升記號，C語言升級版的意思)
 - 以C++和Java語法為基礎
 - 其應用程式是由多個類別(Class)所組成
 - 提供 .NET Framework 類別庫可供使用
 - 可以直接繼承使用視窗元件與架構(類似Java的AWT與Swing framework)，開發視窗介面的程式

C#程式的開發與執行




轉成Native Code(第一次執行時)，載入主記憶體中，由CPU執行

C# 程式開發環境

- 使用微軟的開發工具套件Microsoft Visual Studio進行開發
 - 提供整合開發環境(Integrated Development Environment, IDE)，可用來編輯、編譯、執行和維護應用程式
 - 包括Visual C#、Visual C++、Visual Basic等程式語言開發工具
 - 電腦教室安裝的版本為**Visual Studio 2017 Professional** (學校共同授權版本)
 - 也可以自行下載安裝免費版本的**Visual Studio Community 2022** (官方網站 <https://visualstudio.microsoft.com/> 主頁面請往下捲動到 認識Visual Studio系列 頁面，點選左邊下載Visual Studio，選擇 Community 2022，左下角會出現下載.exe安裝檔。點選此.exe安裝檔執行，執行檔安裝時務必要勾選 .NET桌面開發(使用C#、VB)，右下角點選安裝。需線上註冊一個免費微軟帳號(但有免費試用期限))

認識 Visual Studio 系列




Visual Studio

適用於 Windows 上的 .NET 和 C++ 開發人員的最完善 IDE，全實工具和功能，提升和增強軟體開發的每個階段。

深入了解 →

下載 Visual Studio

- Community 2022
- Professional 2022
- Enterprise 2022



Visual Studio Code

在 Windows、macOS 和 Linux 上執行的獨立原始程式碼編輯器。JavaScript 與 Web 開發人員的首選，具備延伸模組，可支援幾乎任何程式設計語言。

深入了解 →

使用 Visual Studio Code 源代码即同意其 授權 與 隱私聲明

下載 Visual Studio Code

Microsoft | Visual Studio 開發人員工具 下載 購買 訂閱 免費



感謝您下載 Visual Studio

您的下載即將開始，如果您的下載未開始，請[按一下這裡以重試](#)

VisualStudioSetup (1).exe
3.8 MB • 完成




<https://visualstudio.microsoft.com/> 點選Community 2022下載

下載後，點選執行安裝VisualStudioSetup.exe安裝檔


正在安裝 — Visual Studio Community 2022 — 17.1.3

工作負載 個別元件 語言套件 安裝位置


**ASP.NET 與網頁程式開發**
使用 ASP.NET Core、ASP.NET、HTML/JavaScript 及容器 (包括 Docker 支援) 建立 Web 應用程式。


**Python 開發**
對 Python 進行編輯、偵錯、互動式開發及原始檔控制。


**使用 .NET 進行行動開發**
使用 Xamarin 建置 iOS、Android 或 Windows 的跨平台應用程式。這包括將 .NET MAUI 工作負載的預覽作為選用安...

**使用 C++ 的桌面開發**
使用您選擇的工具 (包括 MSVC、Clang、CMake 或 MSBuild)，建置適用於 Windows 的新式 C++ 應用程式。

**Azure 開發**
用於使用 .NET 和 .NET Framework 開發雲端應用程式及建立資源的 Azure SDK、工具及專案。同時包含用於將應用...

**Node.js 開發**
使用非同步的事件驅動 JavaScript 執行階段 Node.js 建置可調整的網路應用程式。

**.NET 桌面開發**
使用 C#、Visual Basic 及 F#，利用 .NET 和 .NET Framework 建置 WPF、Windows Forms 與主控台應用程式。

**通用 Windows 平台開發**
使用 C#、VB 或選用 C++，來建立適用於通用 Windows 平台的應用程式。

安裝詳細資料

▼ .NET 桌面開發

▼ 已包含

- ✓ .NET 桌面開發工具
- ✓ .NET Framework 4.7.2 開發工具
- ✓ C# 與 Visual Basic

▼ 選擇性

- ☒ .NET 的開發工具
- ☒ .NET Framework 4.8 開發工具
- ☒ Blend for Visual Studio
- ☒ Entity Framework 6 工具
- ☒ .NET 分析工具
- ☒ IntelliCode
- ☒ Just-in-Time 偵錯工具
- ☒ Live Share
- ☒ ML.NET Model Builder
- ☐ F# 桌面語言支援
- ☐ PreEmptive Protection - Dotfuscator
- ☐ .NET Framework 4.6.2-4.7.1 開發工具
- ☐ .NET 可攜式程式庫目標套件
- ☐ Windows Communication Foundation
- ☐ SQL Server Express 2019 LocalDB

位置

C:\Program Files\Microsoft Visual Studio\2022\Community 變更...

繼續進行即表示您同意所選 Visual Studio 版本的授權。我們也可讓您使用 Visual Studio 下載其他軟體。此軟體為分開授權，如同[協力廠商聲明](#)或其隨附的授權中所述。繼續進行即表示您也同意該授權。

點選安裝

總共所需空間 9.04 GB

在下載時安裝

安裝(I)

Visual Studio 2019

開啟最近的项目(R)

今天

WindowsFormsApp1.sln 2020/8/13 下午 12:06
C:\Users\user\source\repos\WindowsFormsApp1

安裝完，啟動Visual Studio，
點選建立新的專案(或開啟舊專案)

開始使用

複製存放庫(C)
從像是 GitHub 或 Azure DevOps 等這種線上存放庫取得程式碼

開啟專案或解決方案(P)
開啟本機 Visual Studio 專案或 .sln 檔案

開啟本機資料夾(E)
瀏覽和編輯任何資料夾內的程式碼

建立新的專案(N)
透過程式碼 Scaffolding 選擇專案範本以開始使用

不使用程式碼繼續(U) →

設定新的專案

Windows Forms App (.NET Framework) C# Windows 桌面

專案名稱(N)

WindowsFormsApp2

輸入專案名稱

位置(L)

C:\Users\user\source\repos

設定程式的儲存位置

解決方案名稱(M)

WindowsFormsApp2

☐ 將解決方案與專案置於相同目錄中(D)

架構(I)

.NET Framework 4.7.2

上一步(B)

建立(C)

點選建立

建立新專案

最近使用的專案範本(R)

您最近所存取範本的清單會顯示在這裡。

搜尋範本 (Alt+S)(S)

所有語言(L)

所有平台(P)

所有專案類型(T)

Visual Basic Linux macOS Windows 測試

NUnit 測試專案 (.NET Core)
包含可在 Windows、Linux 和 macOS 上於 .NET Core 執行之 NUnit 測試的專案。
C# Linux macOS Windows 桌面 測試 Web

NUnit 測試專案 (.NET Core)
包含可在 Windows、Linux 和 macOS 上於 .NET Core 執行之 NUnit 測試的專案。
Visual Basic Linux macOS Windows 桌面 測試 Web

Windows Forms App (.NET Framework)
用於建立具有 Windows Forms (WinForms) 使用者介面之應用程式的專案
C# Windows 桌面

WPF 應用程式 (.NET Framework)
Windows Presentation Foundation 用戶端應用程式
C# XAML Windows 桌面

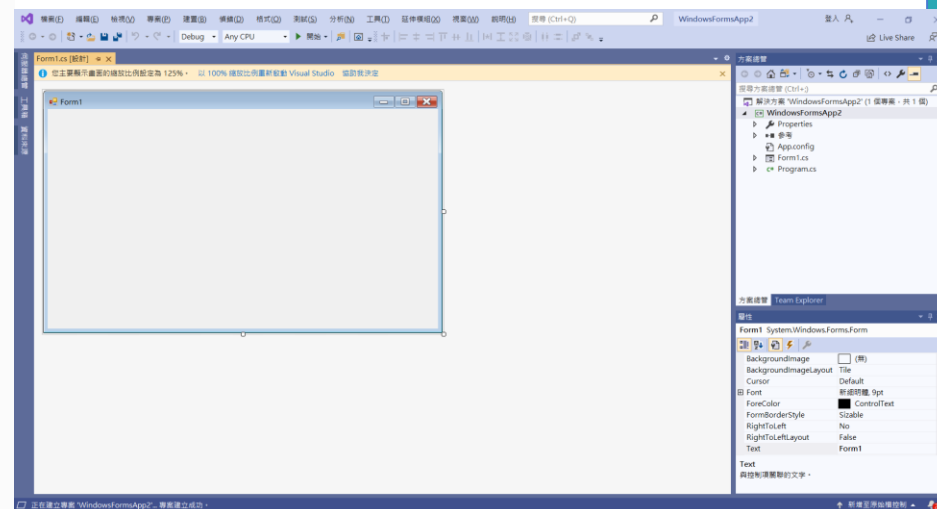
WPF App (.NET Core)
Windows Presentation Foundation 用戶端應用程式
C# XAML Windows 桌面

上一步(B)

下一步(N)

點選下一步

Visual Studio的程式開發介面，開始寫C#程式



C#的基本資料型態(Primitive Data Types)

| 資料型態 | 位元組 | 資料範圍 |
|--------|---------|--|
| bool | 1 byte | true 或 false (布林值) |
| char | 2 bytes | 0~65535 (Unicode) |
| short | 2 bytes | -32,768~32,767 |
| int | 4 bytes | -2,147,483,648~2,147,48,3647 |
| long | 8 bytes | -9,223,372,036,854,775,808~ 9,223,372.036,854,775,807 |
| float | 4 bytes | -3.402823E38~3.402823E38 |
| double | 8 bytes | -1.79769313486232E308~ 1.79769313486232E308 |

基本資料型態的轉換

- 整數常數的預設資料型態為int (佔4個bytes)，浮點數常數的預設資料型態為double (佔8個bytes)
 - 如果要指定常數值的資料型態，需在常數值後面加上型態字元
 - 例如 l或L代表long、f或F代表float
- float x = 1.23;
 - 會出現錯誤訊息，因為double的常數值無法轉換為float資料型態
 - 須改為 float x = 1.23F; 或 double x = 1.23;
- 兩數值作算術運算，會自動將較小範圍的數值轉為較大範圍
 - short → int → long → float → double
- 也可以使用型態轉換運算子(Cast Operator)作強制型態轉換
 - 例如(int) x、(long) y、(double) z、float x = (float) 1.23;

字串資料型態

- string 資料型態

- string str = “Hello World!”;

- 字串可以使用 「 + 」 進行字串的串接運算

- 進行串接運算時，非字串的的運算元會自動轉為字串(但運算元中至少須有一個為字串，若無字串時，可以串接一個空字串“”)

- int n = 100;

- float f = 12.34F;

- string s1, s2=“789”, s3=“34.56”, str = “Hello World!”;

- s1 = “” + n + 123 + “456” + str + f + ‘&’ + n + s2 + s3 + (n + 123);

- 則字串s1的內容為 “100123456Hello World!12.34&10078934.56223”

- 字串轉整數

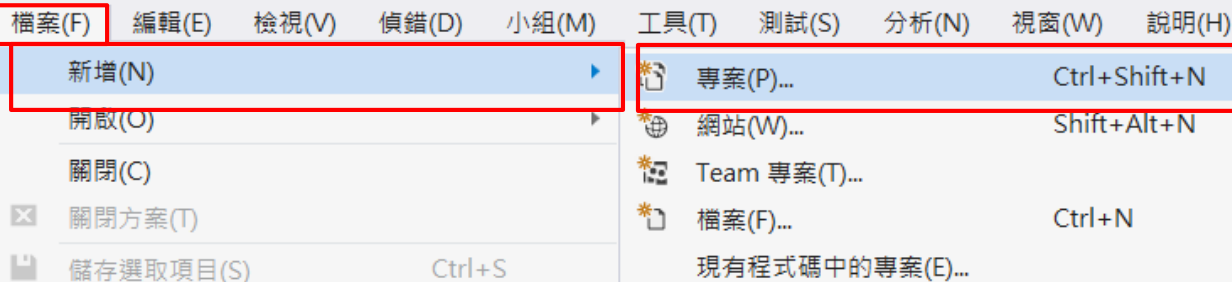
- Convert.ToInt32(s2) 可以將字串”789”轉為32bits (4bytes)的整數789

- 字串轉浮點數

- Convert.ToDouble(s3) 可以將字串”34.56”轉為double的浮點數34.56



起始頁 - Microsoft Visual Studio



新增一個 C# 專案

1. 檔案功能

→ 新增
→ 專案

2. 左邊選取

Visual C#

→ 中間選取

Windows Form

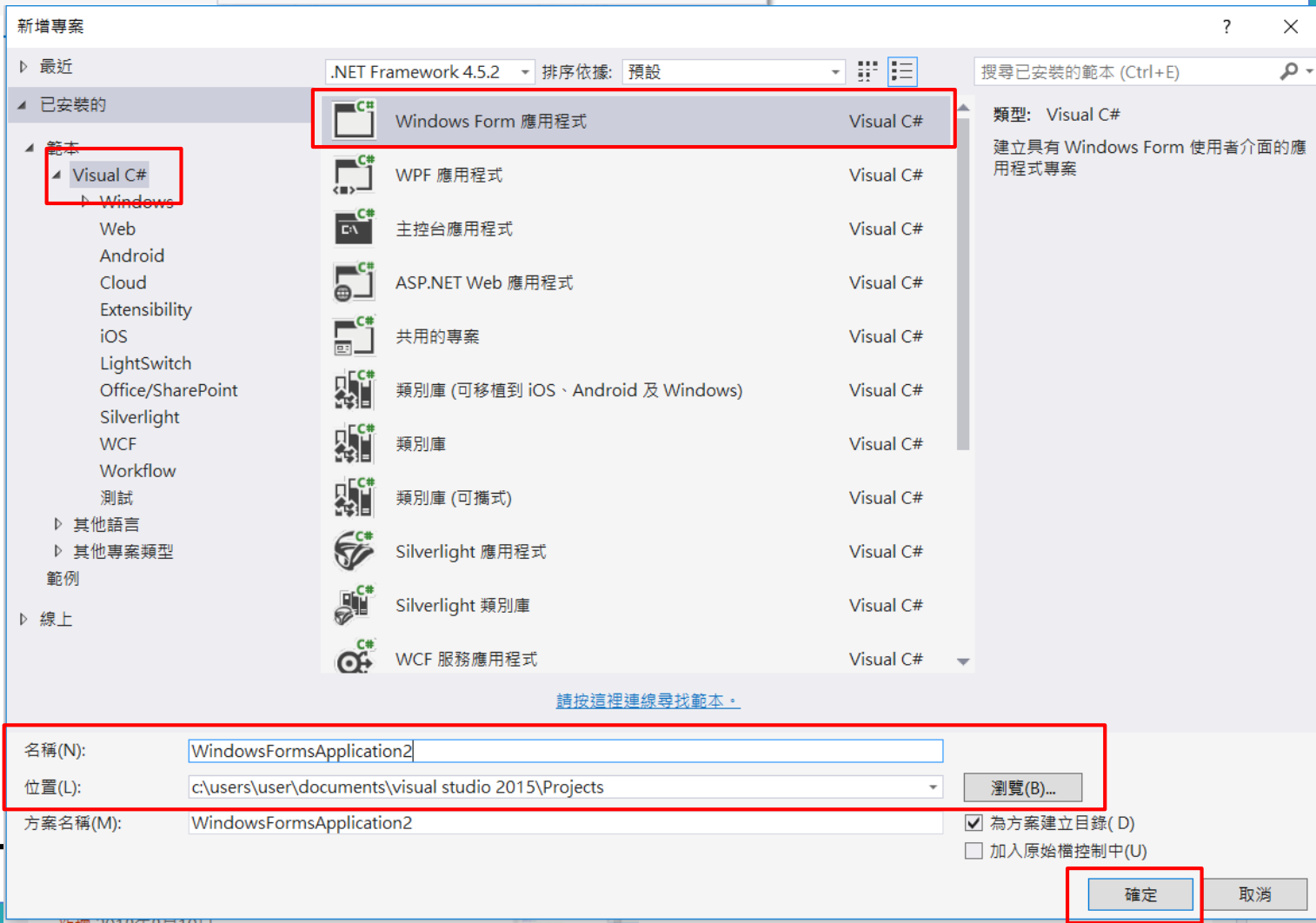
→ 下方輸入

程式名稱

(可瀏覽設定

程式路徑位置)

3. 按確定



Visual C# 的開發編輯環境

WindowsFormsApplication2 - Microsoft Visual Studio

檔案(E) 編輯(E) 檢視(V) 專案(P) 建置(B) 偵錯(D) 小組(M) 格式(O) 工具(I) 測試(S) 分析(N) 視窗(W) 說明(H)

Form1.cs [設計] X

Form1 測試

專案名稱

專案資料夾

自動產生主程式Program類別(Program.cs)和主畫面Form1表單類別(Form1.cs)

可直接拖曳視窗元件到Form1表單中，編輯表單視窗的呈現內容，拉大Form1表單的視窗大小時，屬性列表中的Size屬性值也會跟著改變

屬性列表中可以設定目前所點選的視窗物件(Form1表單物件)之各項屬性值

修改Text屬性的內容值，可以改變Form1表單物件的表單文字顯示(例如把Form1文字改為Form1測試，則表單上的文字顯示也會跟著改變)

方案總管

搜尋方案總管 (Ctrl+;)

方案 'WindowsFormsApplication2' (1 專案)

- WindowsFormsApplication2
 - Properties
 - 參考
 - App.config
 - Form1.cs
 - Program.cs

方案總管 Team Explorer

屬性

Form1 System.Windows.Forms.Form

| | |
|-------------------|------------------------|
| MinimumSize | 0, 0 |
| Opacity | 100% |
| Padding | 0, 0, 0, 0 |
| RightToLeft | No |
| RightToLeftLayout | False |
| ShowIcon | True |
| ShowInTaskbar | True |
| Size | 300, 300 |
| SizeGripStyle | Auto |
| StartPosition | WindowsDefaultLocation |
| Tag | |
| Text | Form1 測試 |
| TopMost | False |

Text
與控制項關聯的文字。

Program類別(Program.cs)

- 點選右邊(方案總管中的Program.cs)可以編輯Program類別
 - C#以namespace來建立類別庫的階層式群組，最上層為System，利用using匯入namespace後，就可以直接使用此名稱空間內的類別或子空間
 - 類別庫的階層式群組以句點連結上下層，例如 System.Windows.Forms
 - Program類別內的Main()為程式起始點，會建立Form1的物件並執行
 - 使用物件(或靜態類別)的屬性或函數也是用句點，例如textBox1物件的Text屬性為textBox1.Text、Application靜態類別的Run()函數為Application.Run()

```
using System;      // 匯入System後就可以直接呼叫Console.WriteLine()，不須寫System.Console.WriteLine()
using System.Windows.Forms;  // 匯入表單及各種控制項類別的namespace
```

```
.....
```

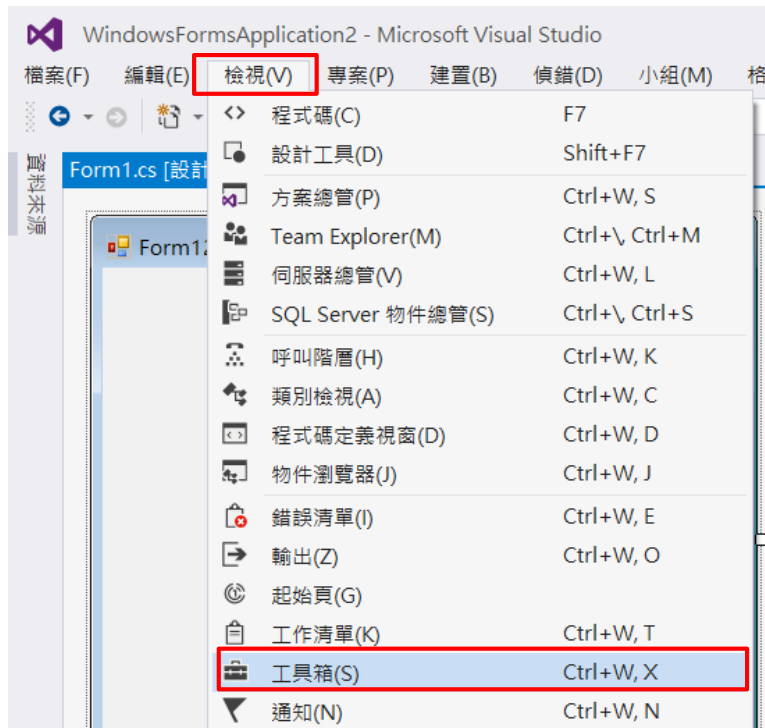
```
namespace 專案名稱      // 定義新增專案的namespace，裡面可以定義多個類別或子空間
{
    static class Program
    {
        .....
        static void Main(String[] args)      // 應用程式的起始點
        {
            .....
            Application.Run( new Form1() );  // new建立Form1表單物件，並執行Form1表單
        }
    }
}
```

Form1表單類別

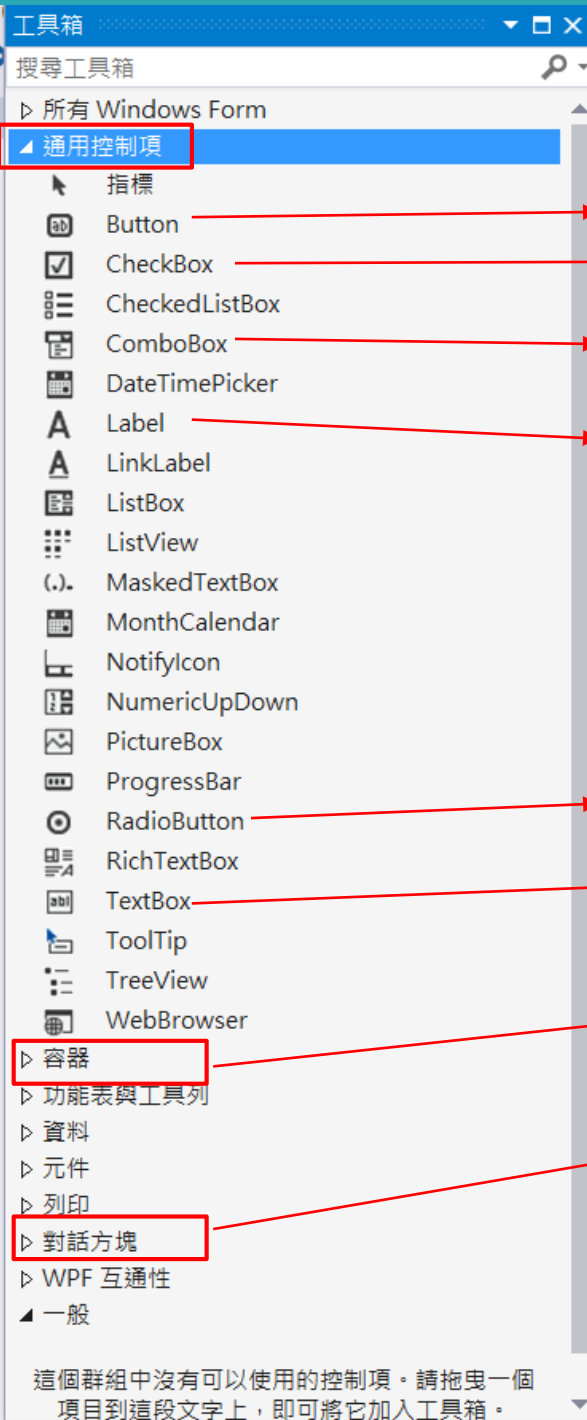
- 點選表單按右鍵(檢視程式碼) 或 點選右邊(方案總管Form1.cs中的Form1) 可以編輯Form1類別
 - **partial class**為部分類別。C#可以把一個類別拆成多個部分類別，編譯時再自動組合成一個完整類別，點選右邊(方案總管Form1.cs中的Form1.Designer.cs)可以看到包含其他控制項物件的Form1部分類別
 - Form1類別**繼承(擴充)**類別庫中的Form類別 (Form1:Form)
 - **InitializeComponent()**進行Form1表單元件的初始化，當Form1表單一被開啟就會執行InitializeComponent()，其程式內容在Form1.Designer.cs

```
using System;
using System.ComponentModel;           // 匯入各種Component元件類別的namespace
using System.Windows.Forms;           // 匯入表單及各種控制項類別的namespace
.....
namespace 專案名稱
{
    public partial class Form1 : Form           // Form1繼承(擴充)Form類別
    {
        public Form1()                       // 與類別同名稱的函數Form1()為建構函數，於new產生物件時執行
        {
            InitializeComponent();           // 執行Form1表單元件的初始化
        }
    }
}
```

視窗介面設計



- 表單 (Form類別所產生的物件)
為各種控制項的容器(container)
1. 檢視功能→開啟工具箱
 2. 點選通用控制項
 3. 拖曳控制項物件到Form1表單中
 4. Form1.Designer.cs 的Form1類別中會自動宣告這些控制項物件



- Button 按鈕
- CheckBox 核取方塊(複選)
- ComboBox 下拉式選單
- Label 標籤(顯示文字)
- RadioButton 選項按鈕(單選)
- TextBox 文字盒(輸入資料)
- 內有GroupBox容器，可以放入多個元件成一個群組
- 內有 OpenFileDialog 讀檔對話方塊
SaveFileDialog 寫檔對話方塊

這個群組中沒有可以使用的控制項。請拖曳一個項目到這段文字上，即可將它加入工具箱。

視窗介面控制項的屬性值設定

ex01-7-18 - Microsoft Visual Studio

檔案(E) 編輯(E) 檢視(V) 專案(P) 建置(B) 偵錯(D) 小組(M) 格式(O) 工具(T) 測試(S) 分析(N) 視窗(W) 說明(H)

Debug Any CPU 開始

Form3.cs [設計]

點選可出現Form表單的視窗設計畫面

點選可出現Form表單的視窗設計畫面

GroupBox容器(可群組多個元件)
CheckBox核取方塊(複選)
RadioButton選項按鈕(單選)
Label標籤(顯示文字)
TextBox文字盒(輸入或顯示資料)
Button按鈕

點擊checkbox1核取方塊，可在右邊視窗中設定其各項屬性值

常用屬性: Name控制項物件的名稱、Text顯示的文字、Font 字型、BackColor背景顏色、ForeColor文字顏色、TextAlign文字對齊方式、MultiLine顯示多行文字、ScrollBar設定捲軸、WordWrap設定自動換行、Visible設定控制項顯示或隱藏、Enabled設定控制項是否啟用、ReadOnly文字盒只顯示不編輯、PasswordChar設定密碼輸入時所顯示的字元

在Text屬性中輸入 雙層吉士堡 \$69，會顯示在表單視窗中

方案總管

搜尋方案總管 (Ctrl+;)

方案 'ex01-7-18' (1 專案)

- ex01-7-18
 - Properties
 - 參考
 - App.config
 - Form1.cs
 - Form2.cs
 - Form3.cs
 - Program.cs

方案總管 Team Explorer

屬性

checkbox1 System.Windows.Forms.CheckBox

| | |
|-------------|------------|
| Locked | False |
| Margin | 3, 2, 3, 2 |
| MaximumSize | 0, 0 |
| MinimumSize | 0, 0 |
| Modifiers | Private |
| Padding | 0, 0, 0, 0 |
| RightToLeft | No |
| Size | 129, 19 |
| TabIndex | 0 |
| TabStop | True |
| Tag | |
| Text | 雙層吉士堡 \$69 |

Text
與控制項關聯的文字。

事件處理程序 (Event Handler)

- 事件驅動(event-driven)的處理方式
 - 當事件發生時(例如按一下按鈕、表單載入、內容修改等)，會觸發執行對應的事件處理程序，若沒有對應事件處理程序，則不執行任何動作
- 建立事件處理程序(event handler)
 - 預設事件：在控制項上按二下，在Form1.cs中的Form1類別中會出現此控制項事件對應的事件處理程序，可寫入對應的執行動作指令

| 控制項 | 預設事件 | 預設的Event handler |
|-----------------|----------------|----------------------------|
| 表單(Form1) | Load | Form1_Load() |
| 按鈕(button1) | Click | button1_Click() |
| 核取方塊(checkBox1) | CheckedChanged | checkBox1_CheckedChanged() |
| 文字盒(textBox1) | TextChanged | textBox1_TextChanged() |

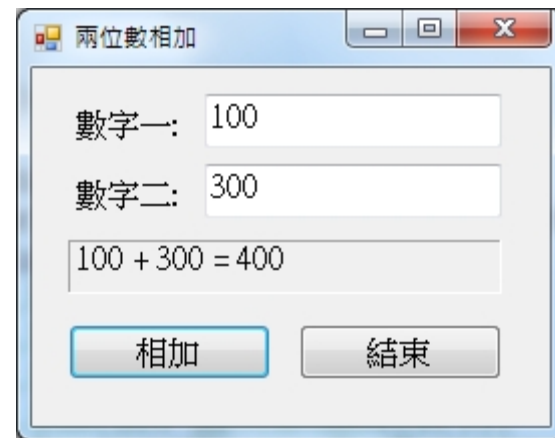
—非預設事件的其他事件：

- 點選控制項→右邊(屬性視窗)→點選閃電圖示(事件)→找到事件→在欄位上按二下，即可出現對應的事件處理程序

程式碼例子

- 題目：輸入兩個數字，顯示相加之後的結果

- Form1 表單物件的Text屬性改為”兩位數相加”
- 加入兩個 **Label** 標籤物件(數字一:、數字二:)
- 加入三個 **TextBox** 文字盒物件(一個ReadOnly)
- 加入兩個 **Button** 按鈕物件(相加、結束)



```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    int n1 = Convert.ToInt32(textBox1.Text);
```

// 因為文字盒中的Text內容為字串，

```
    int n2 = Convert.ToInt32(textBox2.Text);
```

// Convert.ToInt32(字串)可將字串

```
    int result = n1 + n2;
```

// 轉為32位元整數(4 bytes)。

```
    textBox3.Text = n1 + " + " + n2 + " = " + result;
```

// 文字盒中的Text內容指定為

```
}
```

// 將文數字串接後的字串。

```
private void button2_Click(object sender, EventArgs e)
```

```
{
```

```
    this.Close();
```

// this為目前作用中的物件(即Form1表單物件)，

```
}
```

// Close()關閉Form1表單物件後，也結束了應用程式

執行程式碼

- 點選功能表(偵錯→開始偵錯) 或工具列上的▶開始 即可執行
 - 執行檔(.exe)存放於專案目錄中的 \專案名稱\bin\Debug\ 目錄下
 - 假設專案名稱為ex01，儲存的路徑位置為c:\visual studio 2015\Projects，則執行檔在c:\visual studio 2015\Projects\ex01\ex01\bin\Debug\ex01.exe
- 儲存C#專案程式到隨身碟或網路上
 - 新增C#專案時，有設定儲存專案的路徑位置(未設定則為系統預設路徑位置)，到此路徑位置的目錄中，將專案名稱的整個目錄內容複製儲存
- 開啟編輯之前儲存的C#專案程式
 - 點選執行專案目錄中的專案檔(.sln)，例如 \ex01\ex01.sln

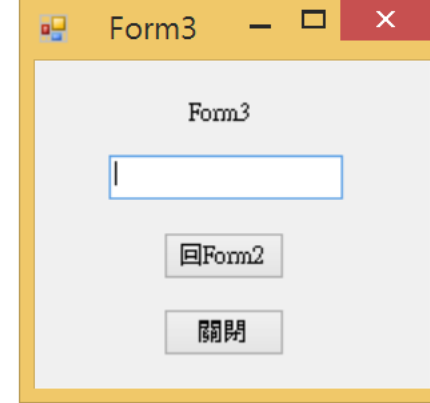
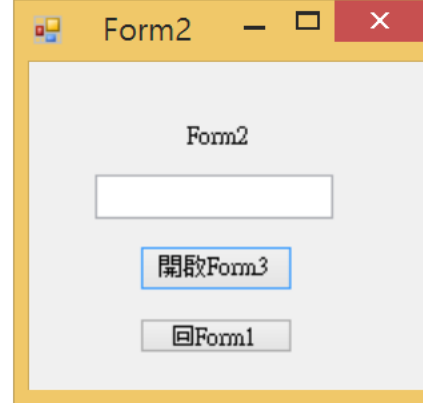
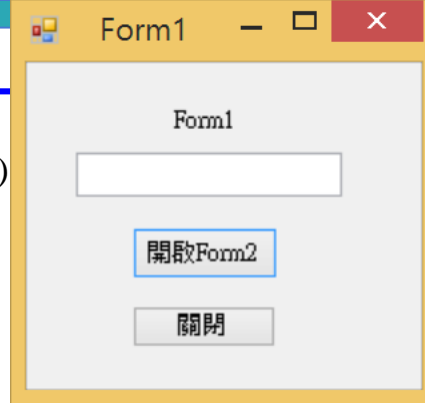
表單切換呼叫

- 新增空白表單
 - 點選右邊(方案總管中的C#專案名稱)→按右鍵(選取加入→Windows Form)→輸入表單名稱

Form1.cs

```
private void button1_Click
(object sender, EventArgs e)
{
    Form2 f = new Form2();
    f.preForm = this;
    f.Show();
    this.Hide();
}

private void button2_Click(object sender, EventArgs e)
{
    this.Close();
}
```



Form2.cs

```
public Form1 preForm;

private void button1_Click(object sender, EventArgs e)
{
    Form3 f = new Form3();
    f.preForm = this;
    f.Show();
    this.Hide();
}

private void button2_Click(object sender, EventArgs e)
{
    this.preForm.Show();
    this.Close();
}
```

Form3.cs

```
public Form2 preForm;

private void button1_Click(object sender, EventArgs e)
{
    this.preForm.Show();
    this.Close();
}

private void button2_Click(object sender, EventArgs e)
{
    this.preForm.preForm.Close();
    this.preForm.Close();
    this.Close();
}
```

選擇控制項

- **GroupBox** 群組方塊(容器): 放在方塊內的選項屬於同一群組
- **CheckBox** 核取方塊(複選): Checked屬性為 true/false
- **RadioButton** 選項按鈕(單選): Checked屬性為 true/false

點餐系統

漢堡/披薩

- ☒ 麥香魚堡 \$59
- ☐ 勁辣雞腿堡 \$69
- ☒ 小披薩 \$259
- ☐ 大披薩 \$429

薯條

- ☒ 薯條
- ☒ 小薯 \$25
- ☐ 大薯 \$35

飲料

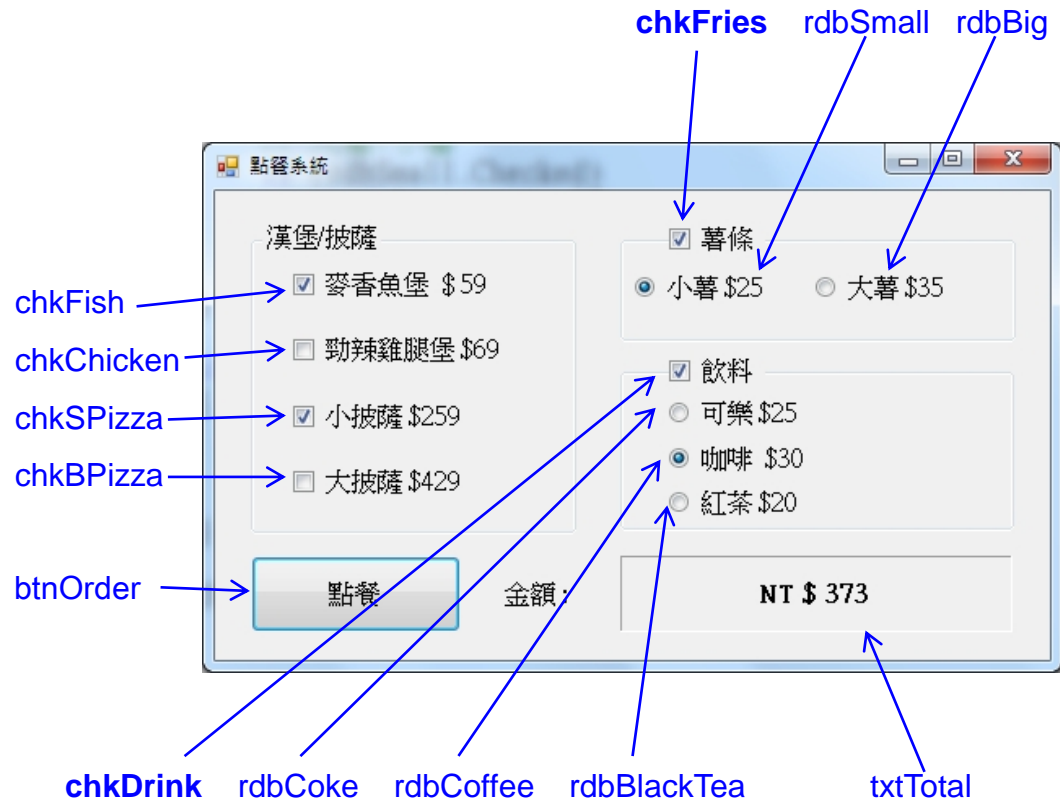
- ☒ 飲料
- ☐ 可樂 \$25
- ☒ 咖啡 \$30
- ☐ 紅茶 \$20

點餐

金額: NT \$ 373

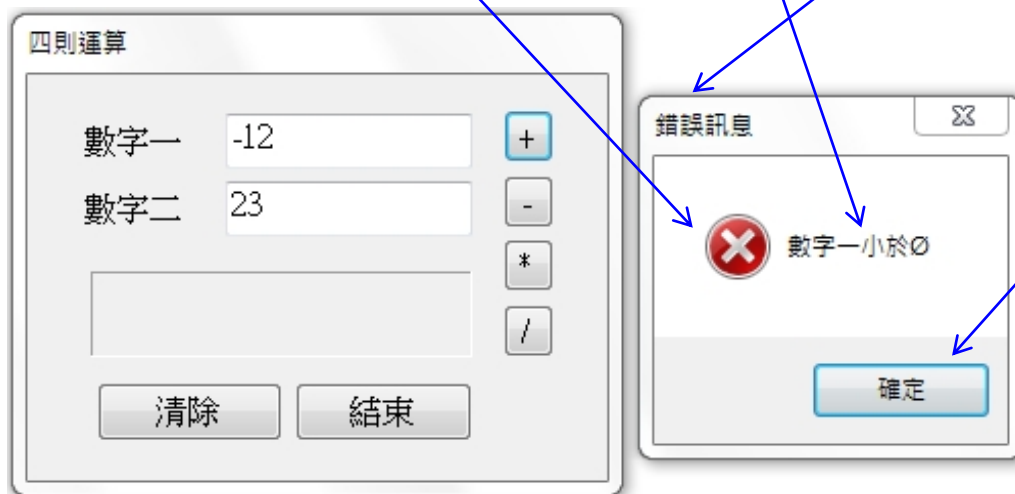
點餐按鈕的程式碼

```
private void btnOrder_Click(object sender, EventArgs e)
{
    int total = 0;
    if (chkFish.Checked) total += 59;
    if (chkChicken.Checked) total += 69;
    if (chkSPizza.Checked) total += 259;
    if (chkBPizza.Checked) total += 429;
    if (chkFries.Checked)
    {
        if (rdbSmall.Checked) total += 25;
        else if (rdbBig.Checked) total += 35;
    }
    if (chkDrink.Checked)
    {
        if (rdbCoke.Checked) total += 25;
        else if (rdbCoffee.Checked) total += 30;
        else if (rdbBlackTea.Checked) total += 20;
    }
    txtTotal.Text = "NT $" + total;
}
```



訊息方塊(MessageBox)

- MessageBox類別可以用來顯示錯誤訊息或輸出執行結果
 - 語法為 **MessageBox.Show(訊息內容, 標題文字, 顯示按鈕, 顯示圖示)**
 - 例如 `MessageBox.Show(“數字一小於0”, “錯誤訊息”, MessageBoxButtons.OK, MessageBoxIcon.Error)`



訊息方塊的顯示按鈕與顯示圖示

- 訊息方塊的顯示按鈕

| MessageBoxButtons 按鈕常數 | 說明 |
|------------------------------------|----------|
| MessageBoxButtons.AbortRetryIgnore | 中止、重試、忽略 |
| MessageBoxButtons.OK | 確定 |
| MessageBoxButtons.OKCancel | 確定、取消 |
| MessageBoxButtons.RetryCancel | 重試、取消 |
| MessageBoxButtons.YesNo | 是、否 |
| MessageBoxButtons.YesNoCancel | 是、否、取消 |

- 訊息方塊的顯示圖示

| MessageBoxIcon 圖示常數 | 說明 |
|--|-------------------|
| MessageBoxIcon.Information、MessageBoxIcon.Asterisk | 藍色圓形內含白色 <i>i</i> |
| MessageBoxIcon.Error、MessageBoxIcon.Stop | 紅色圓形內含白色X |
| MessageBoxIcon.Warning、MessageBoxIcon.Exclamation | 黃色三角形內含黑色！ |
| MessageBoxIcon.Question | 藍色圓形內含白色？ |
| MessageBoxIcon.None | 沒有圖示 |

例子：詢問的對話盒

- `MessageBox.Show()` 會回傳一個型態為 `DialogResult` 的列舉常數值 (**`DialogResult.Yes`** 或 **`DialogResult.No`**)



```
private void button2_Click(object sender, EventArgs e)
{
```

```
    DialogResult result;
```

```
    result = MessageBox.Show("確定結束程式嗎?", "詢問", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
```

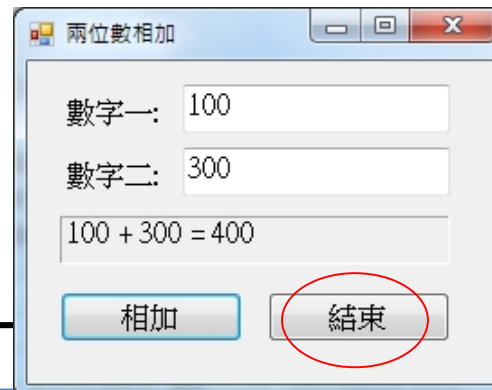
```
    if (result == DialogResult.Yes)
```

```
    {
```

```
        this.Close();
```

```
    }
```

```
}
```



例外處理(Exception Handling)

- C#針對程式執行時，常見的異常執行狀況(例如除以0，資料型態不符等)，已預先定義了一個Exception類別來自動識別和處理，只要把程式碼放入try/catch /finally控制結構中即可
- 使用者也可利用throw new Exception(“自訂錯誤訊息”)丟出自訂的Exception

try

{

// 測試的程式碼(可能會發生已定義的例外狀況或自訂的例外狀況)

}

catch (Exception ex)

{

// 可以有多個Catch程式區塊來捕捉不同的例外狀況

// 例外處理的程式碼

// ex.ToString() 或 ex.Message 都可以取得錯誤訊息的內容

}

finally

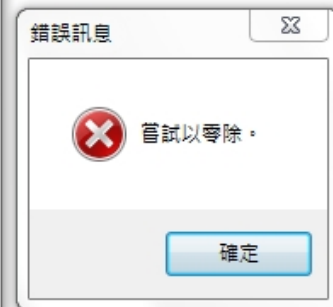
{

// 此區塊之善後程式碼(例如釋放資源等)可以不寫

}

例子：整數的四則運算

- 數字一和數字二須為整數
 - 執行到除以0 (數字二為0)或資料型態不符時會識別出例外狀況
 - 自訂數字一和數字二不可為負數



```
private void button4_Click(object sender, EventArgs e)
```

```
{
```

```
    try
```

```
    {
```

```
        int num1 = Convert.ToInt32(textBox1.Text);
```

```
        if (num1 < 0) throw new Exception("數字一不可為負數");
```

```
        int num2 = Convert.ToInt32(textBox2.Text);
```

```
        if (num2 < 0) throw new Exception("數字二不可為負數");
```

```
        textBox3.Text = num1 + " / " + num2 + " = " + (num1 / num2);
```

```
    }
```

```
    catch (Exception ex)
```

```
    {
```

```
        MessageBox.Show(ex.Message, "錯誤訊息", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
        textBox3.Text = "";
```

```
    }
```

```
}
```

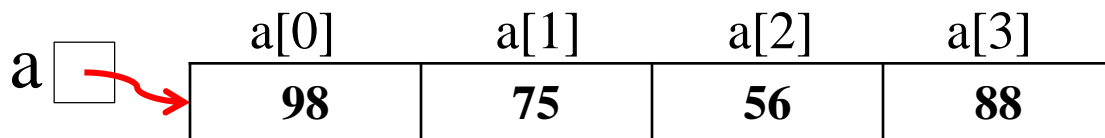


此參數也可以用ex.ToString()

一維陣列

- 一維陣列之宣告與建立

- 資料型態[] 陣列名稱 = new 資料型態 [大小];
- `int[] a = new int[4];`



- 陣列變數a的值是陣列起始的記憶體位址 (間接存取)

- 陣列值之設定

- `a[0] = 98; a[1] = 75; a[2] = 56; a[3] = 88;`

- 建立陣列時，可以同時初始化內容值，有三種寫法

- `int[] a = new int[4] {98, 75, 56, 88};`
- `int[] a = new int[] {98, 75, 56, 88};`
- `int[] a = {98, 75, 56, 88};` // new int[] 可以省略，直接初始內容值

String字串 1/3

- 字串的比較運算子 **==** 和 **!=**
 - 在C#中，字串的內容可以直接比較是否相等
 - `string s1="abcd", s2="abcd", s3="aaaa";`
 - `s1 == s2` 且 `s1 != s3`
- 字串的指定 **=** 和串接 **+**
 - `s3 = s1 + s2 + s3;` // 則s3的內容指定為"abcdabcdaaaa"
- 字串的 **[]**運算子 可以從字串中取出指定位置的字元
 - `string str = "test";`
 - `char c = str[2];` // 則字元c的值為's'
- 可利用**Length**屬性取得字串的長度
 - `str.Length` 的值為4

String字串 2/3

- 字串的比較大小($a < aa < ab < az < ba < bb < bz < z$)
 - `String.Compare(s1,s2)` 會回傳一個整數值
 - `String.Compare(s1,s2)>0` 代表字串s1大於字串s2
 - `String.Compare(s1,s2)==0` 代表字串s1等於字串s2
 - `String.Compare(s1,s2)<0` 代表字串s1小於字串s2
- 子字串的插入與刪除
 - `s1.Insert(整數i,字串s2)` 會將s2字串插入到s1字串中，s1[i]字元的前面
 - `s1 = "hello"; , s2 = "test"; , s1 = s1.Insert(3,s2);`，則s1變成"hel**test**lo"
 - `s1.Remove(整數i,整數n)` 會將s1字串，s1[i]字元開始往後的n個字元刪除
 - `s1 = "hello"; , s1 = s1.Remove(2,2);`，則s1變成"**heo**"
- 空字串""
 - `if (str == "")` 可以判斷是否為空字串

String字串 3/3

- 字串分割

- 字串.Split()

- 會將字串內容依據空格(space)作分割，產生一個字串陣列
 - `string s1 = “abc def ghi jkl mno”;`
 - `string[] a1 = new string[5];`
 - `a1 = s1.Split();`
 - 則 `a1[0]`=“abc”、`a1[1]`=“def”、`a1[2]`=“ghi”、`a1[3]`=“jkl”、`a1[4]`=“mno”

- 字串.Split(',')

- 會將字串內容依據逗號字元(',')作分割，產生一個字串陣列
 - `string s2 = “abc,def,ghi,jkl,mno”;`
 - `string[] a2 = null;` //也可以不指定陣列大小，放入不固定長度的字串陣列
 - `a2 = s2.Split(',');`
 - 則 `a2[0]`=“abc”、`a2[1]`=“def”、`a2[2]`=“ghi”、`a2[3]`=“jkl”、`a2[4]`=“mno”
 - `a2.Length = 5`

二維陣列

- 二維陣列之宣告與建立

- 資料型態[,] 陣列名稱 = new 資料型態 [列數,行數];

- `int[,] s = new int[2,3];`

`s` ↘

| | 0 | 1 | 2 |
|---|----------------------|----------------------|----------------------|
| 0 | <code>s[0, 0]</code> | <code>s[0, 1]</code> | <code>s[0, 2]</code> |
| 1 | <code>s[1, 0]</code> | <code>s[1, 1]</code> | <code>s[1, 2]</code> |

- 建立陣列時，可以同時初始化內容值，有三種寫法

- `int[,] s = new int[2, 3] { {54, 68, 93}, {67, 78, 89} };`

- `int[,] s = new int[,] { {54, 68, 93}, {67, 78, 89} };`

- `int[,] s = { {54, 68, 93}, {67, 78, 89} };`

- 陣列的Length屬性

- `s.Length` 可以取得陣列的元素個數 (任何維度的陣列皆可使用)

檔案處理

- C# 中的檔案以 **串流(Stream)** 的形式呈現，串流代表一個序列裝置，序列裝置以線性方式儲存和讀取資料，此序列裝置可以是檔案、記憶體物件、或網路物件等
- **串流讀取 (StreamReader)** 與 **串流寫入 (StreamWriter)** 為檔案處理類別，用來讀取檔案(串流)和寫入檔案(串流)



- 匯入 **System.IO** 名稱空間 (`using System.IO;`)，就可以使用相關的 I/O 類別來存取檔案系統

純文字檔案(Text File)物件的屬性與方法

- 建立純文字檔案物件

- `FileInfo f = new FileInfo(“路徑+檔名”);`

- Name屬性為檔案名稱
 - FullName屬性為檔案路徑+檔案名稱
 - Extension屬性為副檔名
 - Length屬性為檔案大小
 - Exists屬性為檔案是否存在的布林值(T/F)
 - `OpenText()`開啟現有純文字檔案，並傳回一個StreamReader物件
 - `CreateText()`建立新的純文字檔案，並傳回一個StreamWriter物件

- 記得匯入**System.IO**名稱空間

純文字檔案(Text File)之讀取

- 建立StreamReader串流讀取物件
 - `FileInfo f = new FileInfo(“路徑+檔名”);`
`StreamReader sr = f.OpenText();`
 - 也可以直接 `StreamReader sr = new StreamReader (“路徑+檔名”);`
- 讀取檔案內容
 - `sr.Read();` 利用sr物件，從串流(檔案)讀取一個字元的ASCII碼(整數)，若為串流結尾(檔案結尾)則回傳 -1
 - `char c = (char) sr.Read();` `sr.Read()`所讀取的ASCII碼使用(char)強制轉為字元後再設定給c變數
 - `sr.Peek()` 利用sr物件，檢查串流中下一個字元的ASCII碼(整數)，但並不會讀取出來，若為串流結尾(檔案結尾)則回傳 -1
 - `string s = sr.ReadLine();` 利用sr物件，從串流(檔案)讀取一行，以字串型態回傳給s，若為串流結尾(檔案結尾)則回傳null
 - `string s2 = sr.ReadToEnd();` 利用sr物件，從串流(檔案)讀取所有內容，以字串型態回傳給s2，若無內容則回傳null
- 關閉讀取串流
 - `sr.Close();` 最後記得要關閉讀取串流

純文字檔案(Text File)之寫入

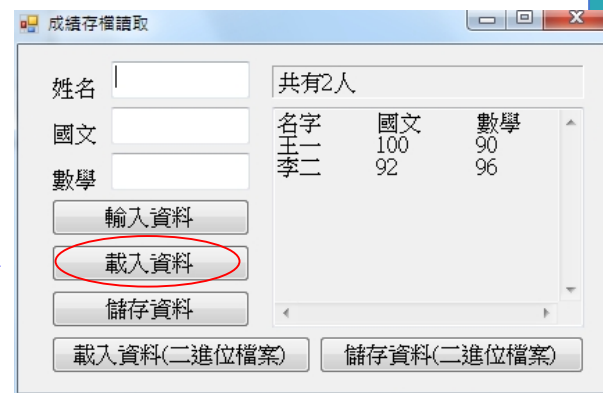
- 建立StreamWriter串流寫入物件
 - `FileInfo f = new FileInfo("路徑+檔名");`
`StreamWriter sw = f.CreateText();`
 - 也可以直接 `StreamWriter sw = new StreamWriter("路徑+檔名");`
- 寫入檔案內容
 - `sw.Write("字串");` 利用串流寫入物件將字串寫入串流(檔案)中
 - `sw.WriteLine("字串");` 利用串流寫入物件將字串及換行寫入串流(檔案)中，也可以寫成`sw.Write("字串"+"r\n");`
- 清除緩衝區資料
 - `sw.Flush();` 資料寫入串流後，記得執行Flush()將資料從緩衝區寫入檔案，並清除緩衝區資料
- 關閉寫入串流
 - `sw.Close();` 最後記得要關閉寫入串流，以免遺失資料

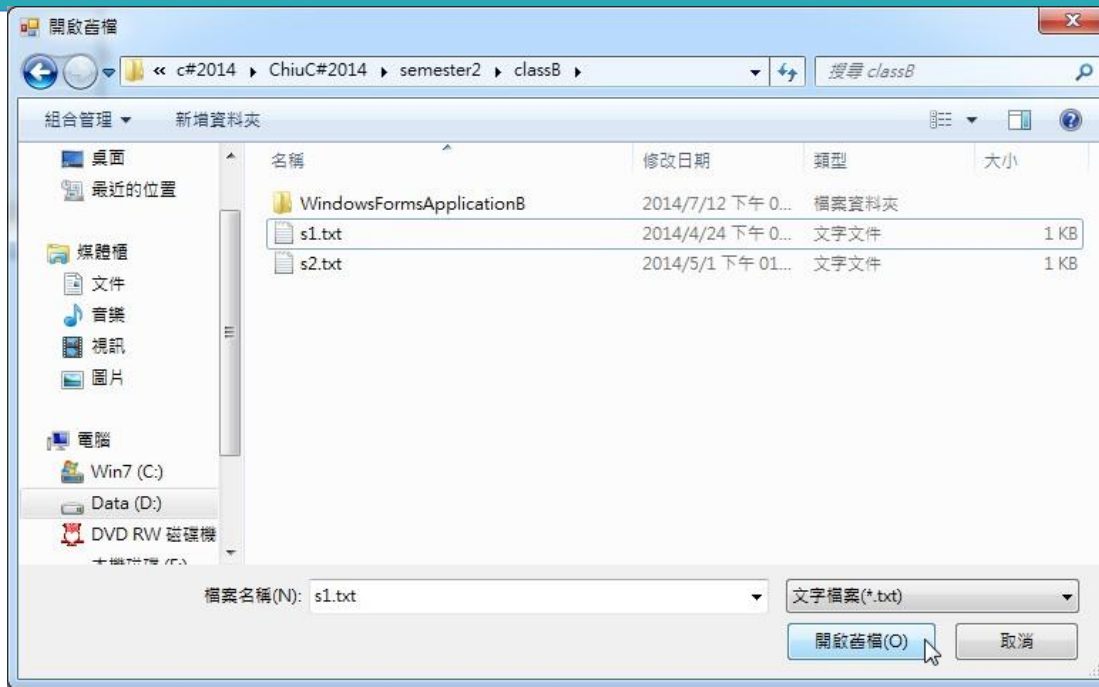
檔案對話方塊(FileDialog)

- 開檔案時由使用者輸入”路徑+檔名”很容易輸入錯誤，檔案對話方塊可以讓使用者選取想要開啟或儲存的檔案
 - 在**工具箱**的**對話方塊**中可以**按兩下**選取**OpenFileDialog**和**SaveFileDialog**
- OpenFileDialog**開啟檔案**之對話方塊、 SaveFileDialog**儲存檔案**之對話方塊
 - **Name**屬性為對話方塊物件名稱、**Title**屬性為對話方塊上的標題文字、**FileName**屬性為所選取的檔案名稱、**InitialDirectory**屬性為對話方塊預設的初始路徑、**Filter**屬性為檔案類型的過濾條件，例如”文字檔案|*.txt”或“所有檔案|*.*”，|的前面為說明文字，|的後面為過濾條件
 - 執行**ShowDialog()**會出現檔案對話方塊，供使用者選取檔案，使用者選取檔案並按”開啟檔案”或”存檔”按鈕後，會回傳**DialogResult.OK**

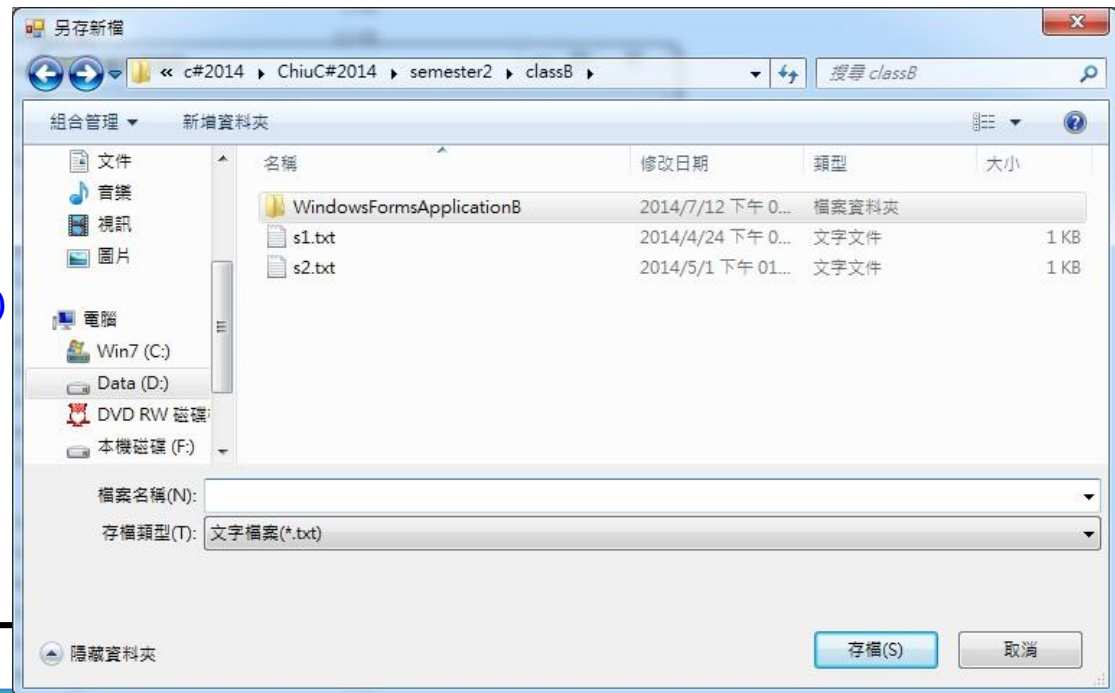
```
private void button2_Click(object sender, EventArgs e)
```

```
{  
    openFileDialog1.Filter = "文字檔案(*.txt)|*.txt"  
    if( openFileDialog1.ShowDialog() == DialogResult.OK )  
    {  
        //產生檔案對話方塊，回傳DialogResult.OK代表使用者已選取檔案  
        FileInfo f = new FileInfo( openFileDialog1.FileName );  
        StreamReader sr = f.OpenText();  
        .....  
        // openFileDialog1.FileName為使用者所選取之檔案
```





`openFileDialog1.ShowDialog()`



`saveFileDialog1.ShowDialog()`

例子:成績處理-資料的輸入、載入和儲存

- 成績資料包含姓名、國文成績、數學成績

- 姓名用一維陣列來儲存
- 成績用二維陣列來儲存

- 可輸入資料後儲存到檔案也可以讀取檔案載入資料

- 配置儲存資料的記憶體

partial class Form1

{

const int MAX_CAPACITY = 50;

string[] name = new string[MAX_CAPACITY];

int [,] scores = new int[MAX_CAPACITY, 2];

int Counter = 0;

}

// 符號常數

// 用一維陣列儲存姓名

// 用二維陣列儲存國文和數學成績

// 目前已儲存成績的學生人數

| 名字 | 國文 | 數學 |
|----|-----|----|
| 王一 | 100 | 90 |
| 李二 | 92 | 96 |

- 輸入姓名及成績資料後，按”輸入資料”按鈕，新增一筆資料

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    if (Counter >= MAX_CAPACITY)
```

```
        MessageBox.Show(“空間已滿”, “錯誤”, MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
    else
```

```
    {
```

```
        name [ Counter ] = textBox1.Text;
```

```
        scores[ Counter, 0 ] = Convert.ToInt32( textBox2.Text );
```

```
        scores[ Counter, 1 ] = Convert.ToInt32( textBox3.Text );
```

```
        Counter++;    // 資料總數加1 (加上新輸入的資料)
```

```
        ShowData();
```

```
    }
```

```
}
```

```
void ShowData()
```

```
{
```

```
    textBox4.Text = “共有” + Counter + ”人”;
```

```
    textBox5.Text = “名字\t國文\t數學\r\n”;
```

```
    for ( int i = 0; i < Counter; i++)
```

```
        textBox5.Text += (name[ i ] + ”\t” + scores[ i, 0 ] + ”\t” + scores[ i, 1 ] + ”\r\n”);
```

```
}
```

| 名字 | 國文 | 數學 |
|----|-----|----|
| 王二 | 100 | 90 |
| 李二 | 92 | 96 |

- 按”儲存資料”按鈕，將所有成績資料寫入檔案

```
private void button3_Click(object sender, EventArgs e)
{
    saveFileDialog1.Filter = "文字檔案(*.txt)|*.txt";
    if ( saveFileDialog1.ShowDialog() == DialogResult.OK )
    {
        FileInfo f = new FileInfo( saveFileDialog1.FileName );
        StreamWriter sw = f.CreateText();
        for( int i = 0; i < Counter; i++ )
        {
            sw.WriteLine( name[ i ] );
            sw.WriteLine( "" + scores[ i, 0 ] );
            sw.WriteLine( "" + scores[ i, 1 ] );
        }
        sw.Flush();
        sw.Close();
    }
}
```

| |
|-----|
| 王一 |
| 100 |
| 90 |
| 李二 |
| 92 |
| 96 |

| 名字 | 國文 | 數學 |
|----|-----|----|
| 王一 | 100 | 90 |
| 李二 | 92 | 96 |

- 按”載入資料”按鈕，讀取檔案中的所有成績資料

```
private void button2_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "文字檔案(*.txt)|*.txt";
    if ( openFileDialog1.ShowDialog() == DialogResult.OK )
    {
        FileInfo f = new FileInfo( openFileDialog1.FileName );
        StreamReader sr = f.OpenText();
        int i = 0;
        while ( sr.Peek() >= 0 )
        {
            name[ i ] = sr.ReadLine();
            scores[ i, 0 ] = Convert.ToInt32( sr.ReadLine() );
            scores[ i, 1 ] = Convert.ToInt32( sr.ReadLine() );
            i++;
        }
        sr.Close();
        Counter = i;
        ShowData();
    }
}
```

| |
|-----|
| 王一 |
| 100 |
| 90 |
| 李二 |
| 92 |
| 96 |

| 名字 | 國文 | 數學 |
|----|-----|----|
| 王一 | 100 | 90 |
| 李二 | 92 | 96 |

二進位檔案(Binary File)的處理

- 純文字檔是以字元為單位進行儲存，二進位檔案則是以byte為單位進行儲存(例如Word文件檔)，因此利用一般純文字編輯器(如記事本)開啟二進位檔案會看到一堆亂碼
- C#需使用FileStream串流物件(二進位串流)來開啟或建立二進位檔案
 - `FileStream fs = new FileStream(“路徑+檔名”, FileMode.模式常數);`
 - FileMode.模式常數可以有
 - FileMode.Open 開啟現有檔案以讀取資料
 - FileMode.Create 寫入覆蓋現有檔案，若檔案不存在則建立新檔案
 - FileMode.Append 寫入到現有檔案的檔尾，若檔案不存在則建立新檔案
 - FileMode.OpenOrCreate 開啟現有檔案讀寫，若檔案不存在則建立新檔案
- 使用BinaryReader物件從二進位串流讀入基本型態的資料
- 使用BinaryWriter物件將基本型態的資料寫入二進位串流
- OpenFileDialog控制項的Filter屬性：“二元檔案(*.dat) | *.dat”

二進位檔案(Binary File)的讀取

- 建立BinaryReader物件

- FileStream fs = new FileStream(“路徑+檔名”, FileMode.Open);
- BinaryReader br = new BinaryReader(fs);

- BinaryReader物件的方法函式

- string s = br.ReadString(); 從開啟的fs串流中讀取一個字串
- char ch = br.ReadChar(); 從開啟的fs串流中讀取一個字元
- int n = br.ReadInt32(); 從開啟的fs串流中讀取一個整數
- double d = br.ReadDouble(); 從開啟的fs串流中讀取double型態的浮點數
- bool b = br.ReadBoolean(); 從開啟的fs串流中讀取一個bool型態的布林值
- br.PeekChar() 檢查fs串流中的下一個字元的ASCII碼(整數)，但並不會讀取出來，若為串流結尾(檔案結尾)則回傳 -1
- br.Close(); 關閉讀取串流

二進位檔案(Binary File)的寫入

- 建立BinaryWriter物件

- FileStream fs = new FileStream(“路徑+檔名”, FileMode.Create);
- BinaryWriter bw = new BinaryWriter(fs);

- BinaryWriter物件的方法函式

- bw.Write(資料); 可以將各種資料型態的資料寫入開啟的fs串流中
- bw.Flush(); 執行 Flush()將資料從緩衝區寫入檔案，並清除緩衝區
- bw.Close(); 關閉寫入串流，以免遺失資料

- 記得匯入System.IO名稱空間(using System.IO;)

- 按”儲存資料(二進位檔案)”按鈕，將成績資料寫入二進位檔案

```
private void button5_Click(object sender, EventArgs e)
```

```
{  
    saveFileDialog1.Filter = "二元檔案(*.dat)|*.dat";  
    if (saveFileDialog1.ShowDialog() == DialogResult.OK )  
    {
```

```
        FileStream fs = new FileStream(saveFileDialog1.FileName, FileMode.Create);
```

```
        BinaryWriter bw = new BinaryWriter( fs );
```

```
        for ( int i = 0; i < counter; i++ )
```

```
        {  
            bw.Write( name[ i ] );  
            bw.Write( scores[ i , 0 ] );  
            bw.Write( scores[ i , 1 ] );  
        }
```

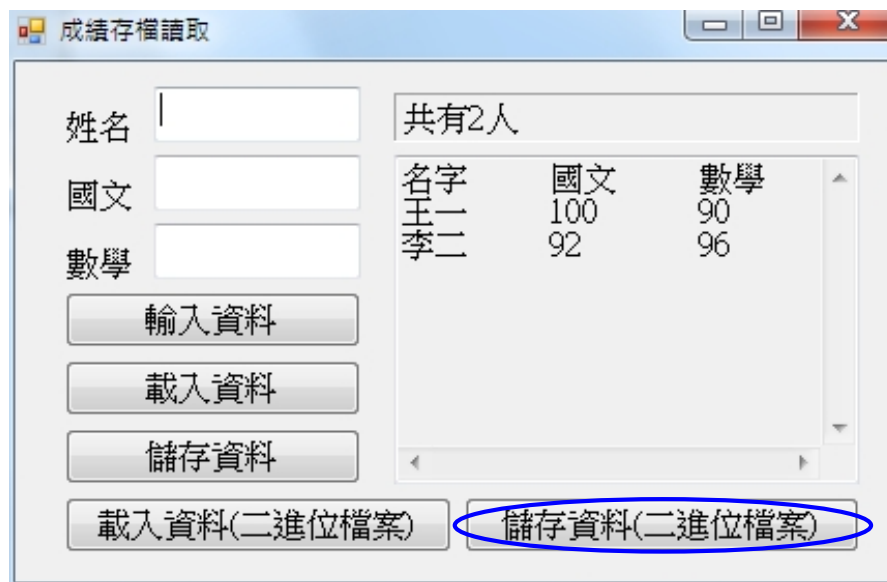
```
        bw.Flush();
```

```
        bw.Close();
```

```
        fs.Close();
```

```
    }
```

```
}
```



```
string[] name = new string[ 50 ];  
int [,] scores = new int[ 50, 2 ];  
int Counter = 0;
```

- 按”載入資料(二進位檔案)”按鈕，讀取二進位檔案的成績資料

```
private void button4_Click(object sender, EventArgs e)
```

```
{
```

```
    openFileDialog1.Filter = "二元檔案(*.dat)|*.dat";
```

```
    if ( openFileDialog1.ShowDialog() == DialogResult.OK )
```

```
    {  
        FileStream fs = new FileStream( openFileDialog1.FileName, FileMode.Open );
```

```
        BinaryReader br = new BinaryReader( fs );
```

```
        int i = 0;
```

```
        while( br.PeekChar() >= 0 )
```

```
        {
```

```
            name[ i ] = br.ReadString();
```

```
            scores[ i , 0 ] = br.ReadInt32();
```

```
            scores[ i , 1 ] = br.ReadInt32();
```

```
            i++;
```

```
        }
```

```
        br.Close();
```

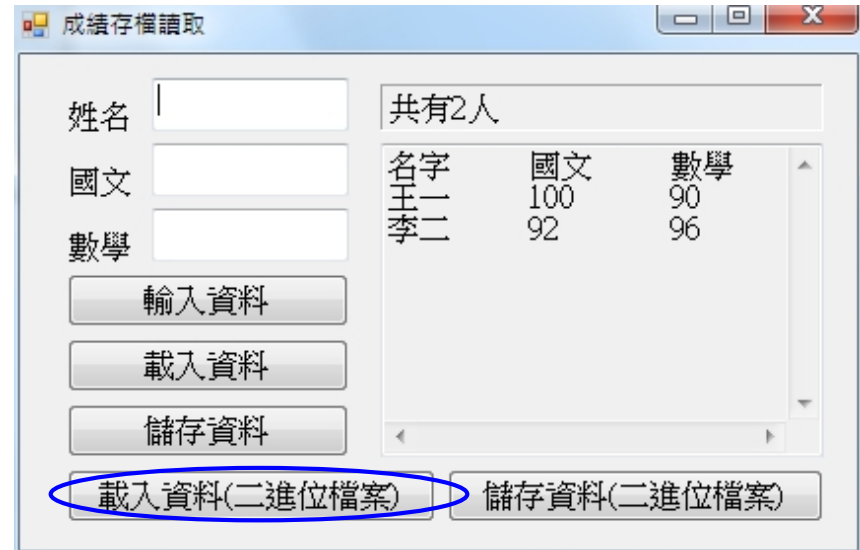
```
        fs.Close();
```

```
        counter = i;
```

```
        ShowData();
```

```
    }
```

```
}
```



函式(方法)參數的傳遞方式 - 傳值呼叫

- **傳值呼叫(Call by value)**: 將變數值複製一份傳給函式的參數，函式內參數值的改變不會變更原來變數的值
- **傳遞陣列**(雖然陣列名稱為陣列的記憶體位址，仍為傳值呼叫)

```
private void button1_Click(object sender, EventArgs e)
{
    int[] s = new int[5];
    s[0] = Convert.ToInt32( textBox1.Text );
    ...
    s[4] = Convert.ToInt32( textBox5.Text );

    int sum = SumArray( s );
    textBox6.Text = "和 =" + sum + "\r\n";
    textBox6.Text += "平均 =" + (sum / 5.0) + "\r\n";
}
```

```
int SumArray( int[] a )
{
    int sum = 0;
    for (int i = 0; i < a.Length; i++)
        sum += a[ i ];
    return sum;
}
```

| 成績 | 值 |
|-----|----|
| 成績一 | 53 |
| 成績二 | 75 |
| 成績三 | 67 |
| 成績四 | 89 |
| 成績五 | 98 |

和 = 382
平均 = 76.4

成績計算

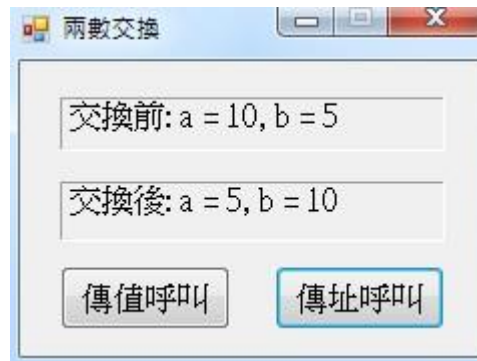
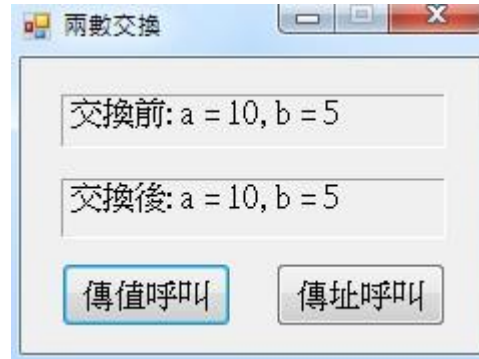
參數傳遞 - 傳址呼叫(call by reference)

- 將變數的位址傳給函式的參數，參數透過此位址來存取變數值，參數值的改變會使得該位址儲存的值改變，而變更原變數的值
- 在參數宣告的前面加上 **ref**，呼叫函式的變數前面也要加上 **ref**
- 例子:交換兩變數的值

(a) call by value (x)

```
void swap(int x, int y)
{
    int t = x;
    x = y;
    y = t;
}

.....
int a = 10, b = 5;
swap(a, b);
.....
```



(b) call by ref (✓)

```
void swap(ref int x, ref int y)
{
    int t = x;
    x = y;
    y = t;
}

.....
int a = 10, b = 5;
swap(ref a, ref b);
.....
```


參數傳遞 - 傳出呼叫 (call by Output)

- 函式可以利用傳出呼叫，一次回傳多個值
- 要回傳值的參數需在宣告的前面加上 **out**，呼叫函式的回傳變數前面也要加上 **out**
- 例子:計算陣列的最大最小值

```
void MaxMinArray(int[] a, out int max, out int min)
{
    max = a[0]; min = a[0];
    for (int i = 1; i < a.Length; i++)
    {
        if (a[i] > max) max = a[i];
        if (a[i] < min) min = a[i];
    }
}

private void button1_Click(object sender, EventArgs e)
{
    int[] s = new int[5];
    s[0] = Convert.ToInt32( textBox1.Text );
    ...
    s[4] = Convert.ToInt32( textBox5.Text );
    int max, min;
    MaxMinArray(s, out max, out min);
    textBox6.Text = "最高分 = " + max + "\r\n";
    textBox6.Text += "最低分 = " + min + "\r\n";
}
```

遞迴(Recursion)

- 使用遞迴解決問題，是將較大的問題不斷分解為較小的子問題，使用相同的解決方法來遞迴呼叫解決 (降階)
- 函式執行中又呼叫自己的函式名稱
- 遞迴函式必須有終止條件以避免無窮遞迴呼叫
- 例如階層函式(factorial)的遞迴呼叫

```
int fact(int n)
```

```
{  
    if (n==0)  
        return(1);  
    else  
        return( n*fact(n-1) );  
}
```

```
int ans = fact(4);
```

fact(4)

4 * fact(3)

3 * fact(2)

2 * fact(1)

1 * fact(0)

return

24

4 * 6

3 * 2

2 * 1

1 * 1

1

fact(4) = 4 * fact(3)

= 4 * (3 * fact(2))

= 4 * 3 * (2 * fact(1))

= 4 * 3 * (2 * (1 * fact(0)))

= 4 * 3 * (2 * (1 * 1))

= 4 * 3 * (2 * 1)

= 4 * 3 * 2

= 4 * 6

= 24

定義類別

- [存取範圍] class 類別名稱 // 沒寫存取範圍時，類別預設為internal

{

// 屬性或方法(函式)的宣告，沒寫存取範圍時預設為private

[存取範圍] [static] 資料型態 變數名稱;

[存取範圍] [static] 傳回值型態 方法名稱(參數1, 參數2, ...)

{

程式碼

[return 值;]

}

}

存取範圍(可視範圍)

| 範圍 | private | internal | protected | public |
|-------------------|---------|----------|-----------|--------|
| 同一類別 | O | O | O | O |
| 同一namespace中的子類別 | | O | O | O |
| 同一namespace，不是子類別 | | O | | O |
| 不同namespace中的子類別 | | | O | O |
| 不同namespace，也非子類別 | | | | O |

- 成員(屬性或方法)的存取範圍不可大於其所屬類別的存取範圍
- 子類別的存取範圍不可大於其父類別的存取範圍
- $\text{private} < \text{internal}, \text{protected} < \text{public}$

例子: 定義一個MyTime類別

- 點選右邊(方案總管中的C#專案名稱)→按右鍵(選取加入→類別)→輸入類別名稱 MyTime.cs
 - 類別沒寫存取範圍(預設為internal)
 - 類別中包含Hour, Minute, Second三個屬性(變數)，沒寫存取範圍(預設為private)

```
namespace ProjectName
{
    class MyTime
    {
        int Hour;
        int Minute;
        int Second;
    }
}
```

- 一個類別可以建立多個物件

```
MyTime t1 = new MyTime();
```

```
MyTime t2 = new MyTime();
```

存取物件中的屬性變數

- 屬性的存取：假設在表單中的按鈕物件要存取t1物件中的屬性

- 物件變數名稱.成員名稱

t1.Hour = 10; // 將 t1 物件中的 Hour 屬性值設定為 10

- 會產生以下的編譯錯誤

‘ProjectName.MyTime.Hour’ 的保護層級導致無法對其進行存取

- 因為成員的預設存取層級為private (只能在同一類別中被存取)

- 將資料成員的存取層級改為public (或internal)，則可以被同一個專案中的類別物件存取 (因為類別為internal，存取層級最高也只會為internal)，但此種方法有風險

(例如，別物件可能
直接將 t1 的 Hour
變數值設定為50這種
錯誤的值，但無法防止)

```
class MyTime
{
    public int Hour;
    public int Minute;
    public int Second;
}
```

類別的封裝性(Encapsulation)

- 保護類別中的資料成員，不要讓存取範圍以外的物件直接更改資料

- 前頁例子中 **public** 的屬性，可能會發生讓其他物件直接修改的風險
- **private** 存取範圍的屬性，可以保護其資料不會被其他物件直接修改
- 其他物件必須透過呼叫此類別物件的 **public 方法(函式)**，才能間接存取 **private** 的資料成員
 - t1.setTime(10,30,40);
 - t1.setTime(**50**,30,40);
- 在 setTime() 中可以找出此錯誤，避免被設定為錯誤資料的風險

```
public void setTime(int h, int m, int s)
{
```

```
    if (h<0 || h>23) MessageBox.Show(“小時的格式不對”, “錯誤”,
        MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
class MyTime
```

```
{
```

```
    private int Hour;
```

```
    private int Minute;
```

```
    private int Second;
```

```
    public string getTime()
```

```
{
```

```
        return Hour + ":" + Minute + ":" + Second;
```

```
}
```

```
    public void setTime(int h, int m, int s)
```

```
{
```

```
        Hour = h;
```

```
        Minute = m;
```

```
        Second = s;
```

```
}
```

```
}
```

類別的建構子(Constructor)

- 建構子是一個類別的**函式(方法)**，建構子名稱必須和類別名稱**相同**，在建立類別的物件時會被呼叫，以設定資料成員的初始值
- 建構子**沒有回傳值(也不加上void)**，通常是宣告為**public**，讓其他物件可以直接呼叫產生此類別的物件來使用
 - (例外) Singleton design pattern為了限制類別的物件只能產生一個，會將建構子宣告為**private**，其他物件必須透過此類別的**另一個public static類別函式**來產生物件，此類別函式中**判斷還未產生過**物件才會呼叫建構子
- 系統會自動預設建構子為 `public MyTime() { }`
- 如果程式中有自行宣告建構子，則C#編譯器就不會自動提供預設建構子
- 建構子支援**多載(Overloading)**
 - 同一個類別中定義兩個以上**同名稱**的函式方法，但傳遞的參數個數或資料型態不同，呼叫函式時會執行參數正確的函式來執行

```
public MyTime(int h, int m, int s) { setTime(h, m, s); }  
public MyTime(int h, int m) { setTime(h, m, 0); }  
public MyTime() { }
```

```
MyTime t1 = new MyTime(9, 30, 50);  
MyTime t2 = new MyTime(21, 40);  
MyTime t3 = new MyTime();
```


類別的建構子(Constructor)

- 建構子是一個類別的**函式(方法)**，建構子名稱必須和類別名稱**相同**，在建立類別的物件時會被呼叫，以設定資料成員的初始值
- 建構子**沒有回傳值(也不加上void)**，通常是宣告為**public**，讓其他物件可以直接呼叫產生此類別的物件來使用

—(例外) Singleton design pattern為了限制類別的物件只能產生一個，會將建構子宣告為**private**，其他物件必須透過此類別的**另一個public static類別函式**來產生物件，此類別函式中**判斷還未產生過物件**才會呼叫建構子

- 系統會自

- 如果程式

- 建構子支

—同一個

料型態

public MyTime

public MyTime

public MyTime

```
public class MyTime
{
    private static MyTime uniqueInstance ;
    public static MyTime getInstance()
    {
        if (uniqueInstance == null)
            uniqueInstance = new MyTime() ;
        return uniqueInstance ;
    }
    private MyTime( ) { ... }
}
```

設建構子

數個數或資

```
time(9, 30, 50);
time(21, 40);
time();
```

物件陣列

- 多個同類別的物件所構成的陣列

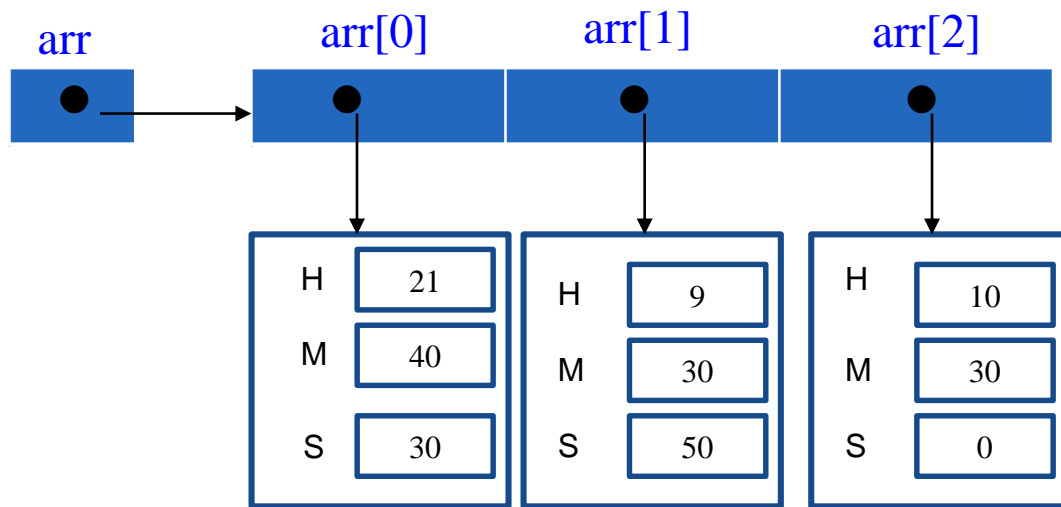
```
MyTime[] arr = new MyTime[3];
```

```
arr[0] = new MyTime( );
```

```
arr[0].setTime(21, 40, 30);
```

```
arr[1] = new MyTime(9, 30, 50);
```

```
arr[2] = new MyTime(10, 30, 0);
```



物件串列(泛型集合Generic Collection)

- 物件陣列中儲存的物件個數為固定不變，物件串列(泛型集合)為物件的鏈結串列，儲存的物件個數為可變的，可以動態地加入或刪除同一類別的物件
- 使用物件串列(泛型集合)需匯入 **using System.Collections.Generic;**
- 語法: **List<物件類別>** 串列變數 = new **List<物件類別>** ();
- 使用泛型集合的[]運算子 可以從串列中取出指定位置的物件
 - 串列變數[i] 可以取出串列中的第i個物件(從0開始)
- 物件串列(泛型集合)常用的方法函式
 - **Clear()** 刪除串列中的所有物件
 - **Add(物件變數)** 加入物件到串列中
 - **Remove(物件變數)** 從串列中刪除物件
 - **RemoveAt(整數i)** 從串列中刪除第i個物件(串列變數[i])
 - **IndexOf(物件變數)** 回傳物件變數在串列中的index (從0開始)
- 物件串列(泛型集合)的屬性
 - **Count** 目前串列中的物件個數

建立一個MyTime物件的串列list1

```
List<MyTime> list1 = new List<MyTime>();  
list1.Add(t1);  
list1.Add(t2);  
list1[0].setTime(11,20,30);
```

靜態成員 (static members 屬於類別本身的成員)

- 一般的物件成員，包括物件屬性和物件方法，都需要產生物件後才能使用
- 可以利用static宣告靜態成員，為類別本身的成員，不需要產生物件即可使用，可以將一些不會因個別物件而有所差異的成員宣告為靜態
 - 類別成員的宣告若加上static則為靜態成員，未加static為物件成員
 - 靜態屬性變數的記憶體僅有一份，由該類別所產生的所有物件共享，所以無法以this(目前物件)來使用靜態屬性變數
 - 靜態方法函式屬於類別本身，函式內也無法使用this來參考到物件
 - 實體成員的存取
 - 物件名稱.實體成員名稱
 - 靜態成員的存取
 - 類別名稱.靜態成員名稱 (如：Convert.ToInt32(“567”))

靜態成員的例子: MyTime物件的個數

```
class MyTime
```

```
{  
    private static int count=0;  
    private int Hour;  
    private int Minute;  
    private int Second;  
    public MyTime() { count++; }  
    public static int getCount() { return count; }  
    public string getTime()  
    {  
        return Hour + ":" + Minute + ":" + Second;  
    }  
    public void setTime(int h, int m, int s)  
    {  
        if (h<0 || h>23) MessageBox.Show("小時的格式不對", "錯誤", ...);  
        Hour = h;  
        Minute = m;  
        Second = s;  
    }  
}
```

```
MyTime t1 = new MyTime();  
t1.setTime(11,30,0);  
MyTime t2 = new MyTime();  
t2.setTime(18,10,30);  
MyTime t3 = new MyTime();  
t3.setTime(7,50,20);  
string str;  
str = "總共產生" + MyTime.getCount() +  
      "個MyTime物件\r\n";
```

表單切換

● 新增表單

— 點選右邊(方案總管中的C#專案名稱)→按右鍵(選取加入→Windows Form)→輸入表單名稱(Form2.cs)

● 非強制回應(Modeless)表單

1) 非”啟動表單”的開啟

```
Form2 f2 = new Form2();
```

```
f2.Show();    //不會有回傳值，後面的程式碼仍會繼續執行
```

2) 關閉表單

```
f2.Close();    // Close()關閉後就不能再呼叫Show()
```

3) 隱藏表單

```
f2.Hide();    // 表單隱藏，但可以再呼叫Show()來顯示表單
```

強制回應表單(Modal)

- 對話方塊，使用者必須輸入資料和關閉視窗後，才能繼續執行應用程式。
例如：MessageBox.Show()所建立的訊息方塊

- 一般表單若使用.ShowDialog()方法來開啟，即須強制回應

```
Form3 f3 = new Form3();
```

```
f3.ShowDialog(); // 後面的程式碼須等對話方塊關閉後才會繼續執行
```

```
-----
```

- 判斷使用者在表單上執行的動作後，會回傳一個DialogResult屬性值
 - 可以在強制回應表單的按鈕控制項上，選取設定DialogResult屬性的值 (None, OK, Cancel, Abort, Retry, Ignore, Yes, No)，代表按此按鈕後會回傳的DialogResult屬性值，只有預設的None值不會關閉對話方塊

```
if (f3.DialogResult == DialogResult.OK)
```

```
{
```

```
-----
```

```
}
```

- if (f3.ShowDialog() == DialogResult.OK) { ----- } 將開啟表單和判斷回應兩個動作寫在一起也可以

表單切換的例子

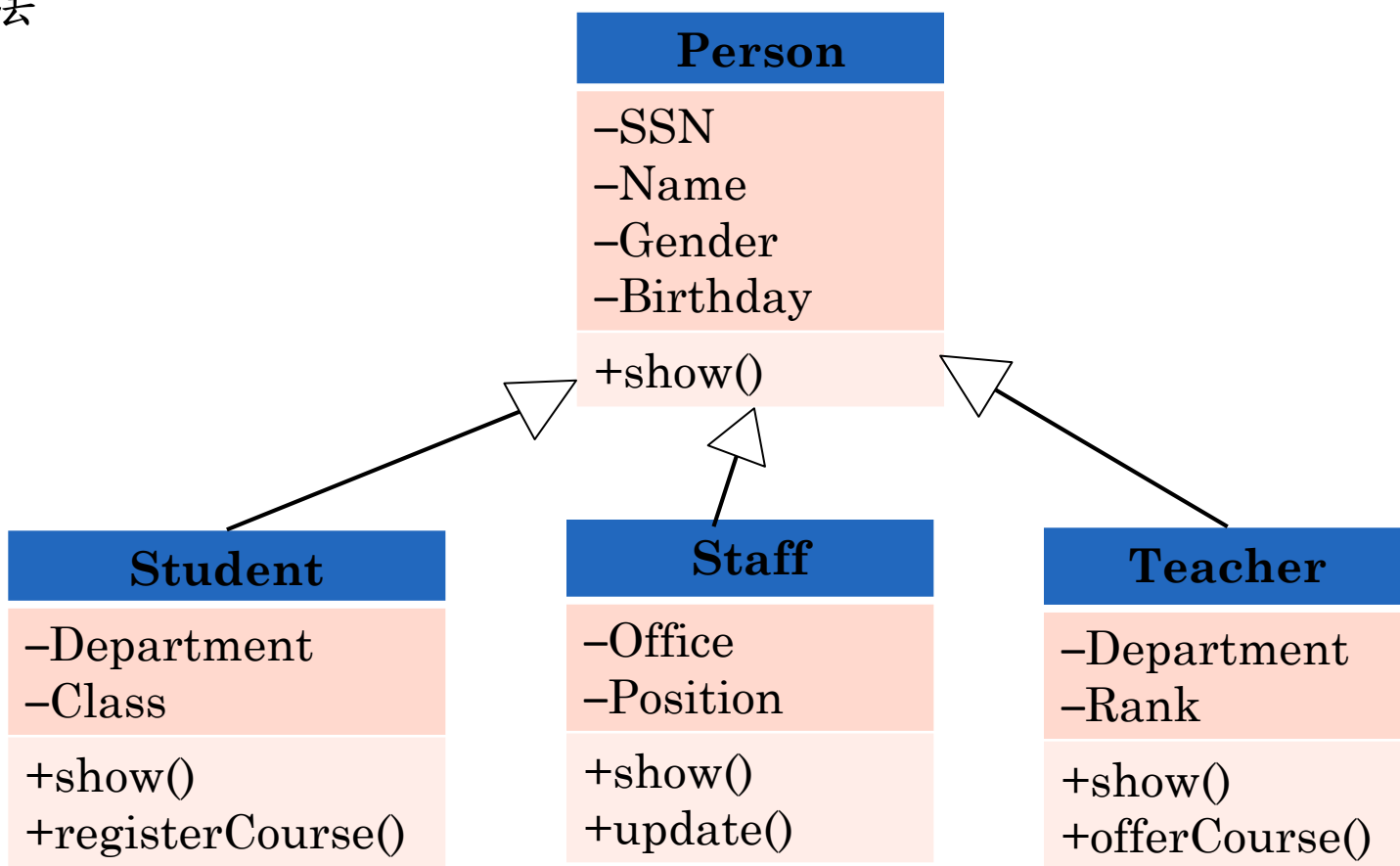
```
private void button1_Click(object sender, EventArgs e)
{
    Form2 f2 = new Form2();
    f2.ShowDialog();
    if (f2.DialogResult == DialogResult.OK)
        MessageBox.Show("你輸入了" + f2.getInput() + "!");
    if (f2.DialogResult == DialogResult.Cancel)
        MessageBox.Show("你按了取消鍵");
}
```

```
partial class Form2
{
    public string getInput()
    {
        return textBox1.Text;
    }
}
```



類別的繼承(Inheritance)

- 當定義新類別時，可以直接**繼承現有類別**以重複利用(reuse)其屬性變數和方法函式，並依新類別的需要**增加**或**改寫(Override)**繼承類別的屬性和方法
- 舉例來說，在校務管理系統中，Student類別、Staff類別、和Teacher類別都繼承Person類別的許多共同的屬性和方法，但各自也會具有其獨有的屬性和方法



類別繼承的語法

[存取範圍] class 子類別名稱 : 父類別名稱

{

// 屬性變數和方法函式

}

- C#只允許單一繼承，**不允許多重繼承**(多個父類別)
- 沒有指定父類別時，C#編譯器將自動以**System.Object**為其父類別，System.Object是整個類別階層體系的根類別(Root)
- 類別預設的存取範圍是**internal**，若沒寫存取範圍就是internal
- **子類別**的存取範圍不能大於**父類別**的存取範圍
 - internal class **可以繼承** public class，但 public class **不能繼承** internal class

| [合法] | [不合法] |
|---|---|
| <pre>public class MyBase { //class members } internal class MyClass : MyBase { // class members }</pre> | <pre>internal class MyBase { //class members } public class MyClass : MyBase { // class members }</pre> |

屬性變數和方法函式的存取範圍

- 屬性變數和方法函式預設的存取範圍為**private**，若沒寫存取範圍就是private
 - － **屬性和方法**的存取範圍不能大於其**所屬類別**的存取範圍

namespace A

```
{
    public class A1
    {
        protected int p ;
        ---
    }
    class A2:A1
    {
        //可直接存取A1的p
    }
    class A3
    {
        //無法直接存取A1的p
    }
}
```

namespace B

```
{
    class B1:A1
    {
        //可直接存取A1的p
    }
    class B2
    {
        //無法直接存取A1的p
    }
}
```

例子: Cylinder類別繼承Circle類別

Circle

#radius: int

+Circle()
+Circle(int)
+getRadius(): int
+getArea(): double



Cylinder

-length : int

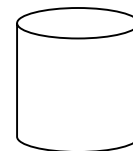
+Cylinder(int, int)
+getLength(): int
+getArea(): double

Class Cylinder : Circle

```
{  
    int length;  
    public Cylinder(int r, int l)  
    { radius = r; length = l; }  
    public int getLength()  
    { return length; }  
    public new double getArea()  
    {  
        double circleArea = Math.PI * radius * radius;  
        double circumference = 2 * Math.PI * radius;  
        return 2 * circleArea + circumference * length;  
    }  
}
```

Class Circle

```
{  
    protected int radius;  
    public Circle() { }  
    public Circle(int r)  
    { radius = r; }  
    public int getRadius()  
    { return radius; }  
    public double getArea()  
    { return Math.PI*radius*radius; }  
}
```



1. 要改寫父類別中同名稱方法的宣告，資料型態前要加new
2. 產生子類別的物件時，先執行父類別的預設建構子Circle()，才會執行子類別的建構子Cylinder(5,10)

```
Cylinder cyl = new Cylinder(5, 10);  
MessageBox.Show("圓柱體cyl的半徑 = " + cyl.getRadius() +  
    "\r\n" + "圓柱體cyl的高度 = " + cyl.getLength() + "\r\n" +  
    "圓柱體cyl的表面積 = " + cyl.getArea(), "Cylinder物件",  
    MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Cylinder物件



圓柱體cyl1的半徑 = 5
圓柱體cyl1的高度 = 10
圓柱體cyl1的表面積 = 471.238898038469

確定

物件的型態轉換(Type Casting)

- 子類別(Cylinder)的物件可以由父類別(Circle)的變數來參考(當作父類別的物件來使用) => 隱含的型態轉換 (物件型態從Cylinder轉換成Circle的物件)
 - Circle c = new Cylinder();
 - 物件變數c宣告為類別Circle的物件，代表它可以看到和用到類別Circle的屬性和方法，而子類別Cylinder的物件足以提供這些成員
- 但是父類別(Circle)的物件則不能當作子類別(Cylinder)的物件來使用
 - Cylinder cy = new Circle(); 為錯誤的
 - 物件變數cy宣告為類別Cylinder的物件，但是父類別Circle的物件無法提供類別Cylinder的屬性和方法
 - Cylinder cy = c; 也是錯誤的 (即使前面宣告 Circle c = new Cylinder();)
 - 因為c宣告為類別Circle的物件，為父類別的物件
 - Cylinder cy = (Cylinder) c; 則為正確的 => 明顯的型態轉換
 - 利用物件型態轉換運算子(Cylinder)將 c 的物件型態從父類別Circle的物件轉換回子類別Cylinder的物件

靜態繫結(static binding)與動態繫結(dynamic binding)

- 子類別使用 **new** 方法來改寫父類別中的 **同名稱方法** 為 **static binding**，在 **編譯時期(compile)** 就會根據物件的類別決定要呼叫的同名方法是父類別或子類別的
 - public **new** double getArea() { }
- 子類別使用 **override** 方法來改寫父類別中 **同名稱方法** 為 **dynamic binding**，是在 **執行時期(run)** 才依當時物件實際所參考的類別來呼叫父類別或子類別的方法
 - 當一個父類別有 **多個子類別**，而每個子類別各自對於同名稱方法有不同的改寫函式，若 **物件的產生要到執行時才知道** 是哪個子類別的物件，編譯時期無法事先得知時，則需使用 **override** 方法來改寫
 - 父類別的同名稱方法要加上 **virtual**，子類別的同名稱方法要加上 **override**

Class Circle

```
{
    .....
    public virtual double getArea()
    {
        return Math.PI*radius*radius;
    }
}
```

Class Cylinder : Circle

```
{
    public override double getArea()
    {
        double circleArea = Math.PI * radius * radius;
        double circumference = 2 * Math.PI * radius;
        return 2 * circleArea + circumference * length;
    }
}
```

抽象類別(abstract class)

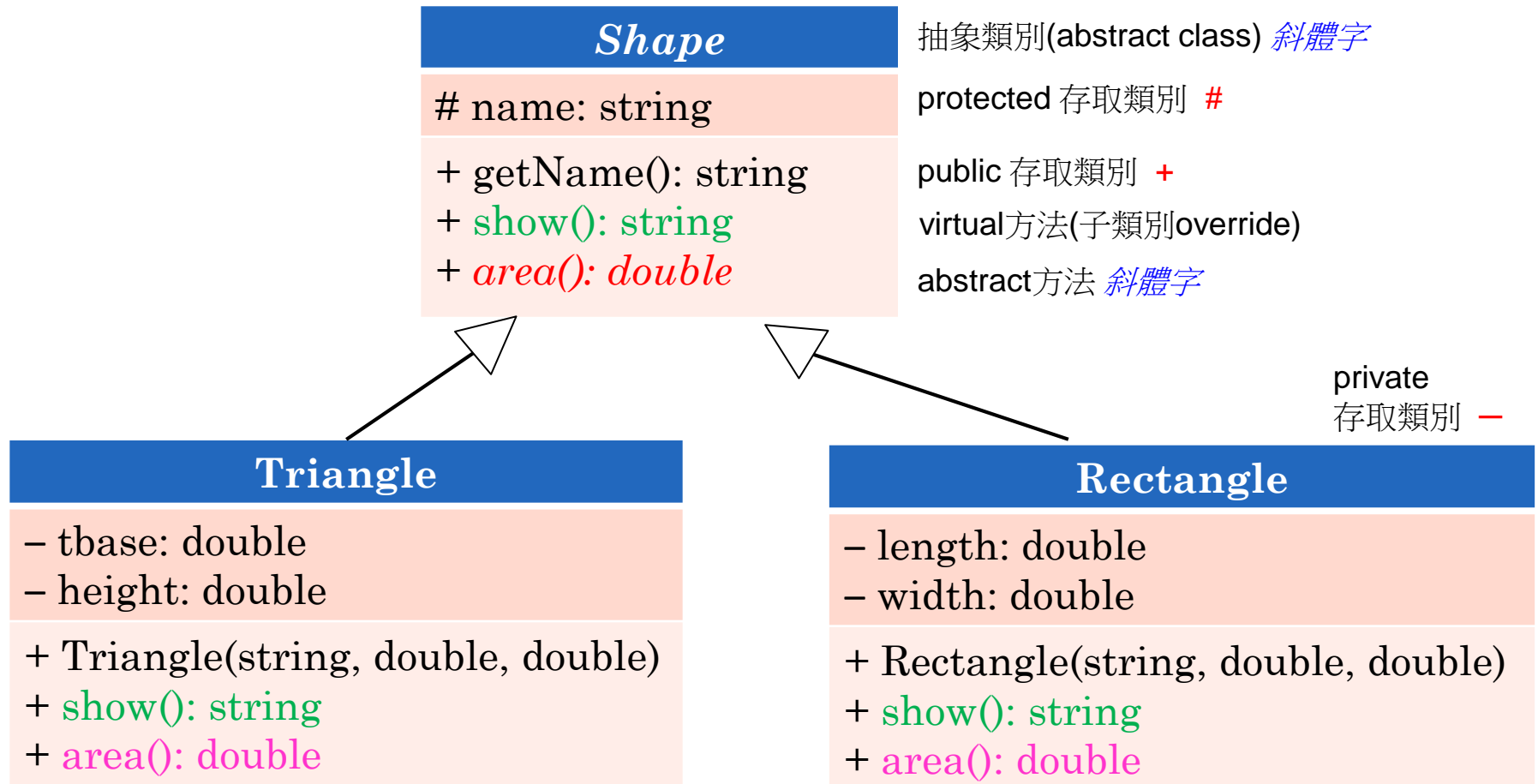
● 抽象類別 (abstract class)

- 宣告抽象類別時，要加上abstract
- 抽象類別不能產生物件(不能new物件，但可以宣告為物件變數的類別)，因此只能作為父類別，讓其他子類別所繼承
- 例如：形狀 (Shape) 是一種共通的抽象類別，而其子類別三角形 (Triangle) 和矩形 (Rectangle) 才能產生物件(具體的形狀)

● 抽象方法 (abstract method)

- 宣告抽象方法時，要加上abstract
- 含有抽象方法的類別必須宣告為抽象類別
- 抽象方法是一種虛擬方法(不用再宣告Virtual)，不必提供函式內容
- 繼承抽象類別的子類別中，必須搭配override來實作抽象方法
- 例如：形狀 (Shape) 有一個area()方法來取得形狀 (Shape) 的面積，但必須具體知道是三角形或矩形，才能真正計算其面積。
 - 因此，可以在Shape類別中，將area()宣告為抽象方法，再由其子類別Triangle類別和Rectangle類別來實作area()的函式內容

抽象類別與抽象方法的例子(UML類別圖)



Shape抽象類別

```
abstract class Shape
```

```
{  
    protected string name;  
    public string getName()  
    {  
        return name;  
    }  
    public virtual string show()  
    {  
        return name;  
    }  
    public abstract double area();  
}
```

```
class Triangle : Shape
```

```
{  
    private double tbase;  
    private double height;  
    public Triangle(string n, double b, double h)  
    {  
        name = n;  
        tbase = b;  
        height = h;  
    }  
    public override string show()  
    {  
        return "Triangle: " + name + "(" + tbase + "," + height + ")";  
    }  
    public override double area()  
    {  
        return 0.5 * tbase * height;  
    }  
}
```

Rectangle子類別

```
class Rectangle : Shape
```

```
{  
    private double length;  
    private double width;  
    public Rectangle(string n, double l, double w)  
    {  
        name = n;  
        length = l;  
        width = w;  
    }  
    public override string show()  
    {  
        return "Rectangle: "+name+"("+length+", "+width+")";  
    }  
    public override double area()  
    {  
        return length * width;  
    }  
}
```



新增Triangle

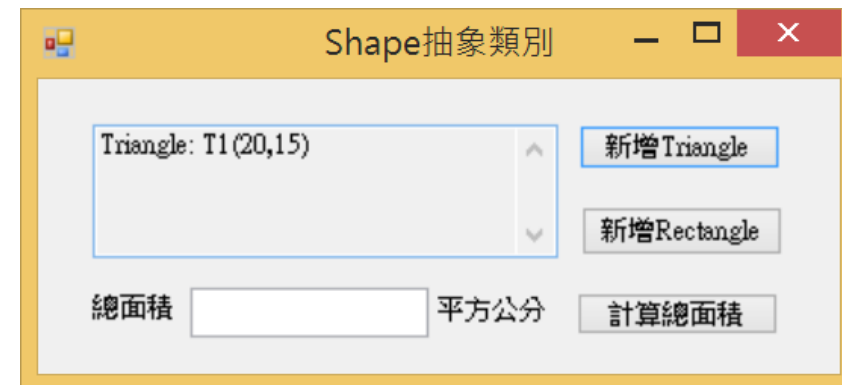
名稱 T1

底 20

高 15

確定

取消



Shape抽象類別

Triangle: T1(20,15)

新增Triangle

新增Rectangle

總面積 平方公分

計算總面積



Shape抽象類別

Triangle: T1(20,15) Rectangle: R1(6,12) Rectangle: R2(10,7) Triangle: T2(9,21) Triangle: T3(11,2)

新增Triangle

新增Rectangle

總面積 397.5 平方公分

計算總面積

新增Form2表單來輸入資料 (Form3輸入Rectangle物件)

```
partial class Form1 : Form
```

```
{  
    List<Shape> allShape = new List<Shape>();
```

```
    private void button1_Click(object sender, EventArgs e)
```

```
{  
        try  
        {  
            Form2 f2 = new Form2();  
            f2.ShowDialog();  
            string newName;  
            double tbase, height;  
            if (f2.DialogResult == DialogResult.OK) //Form2按確定鍵  
            {  
                f2.getData(out newName, out tbase, out height);  
                Triangle x = new Triangle(newName, tbase, height);  
                allShape.Add(x); //將Triangle物件加入Shape串列中  
                showShape();  
            }  
        }  
        catch (Exception ex)  
        {  
            MessageBox.Show(...);  
        }  
    }
```


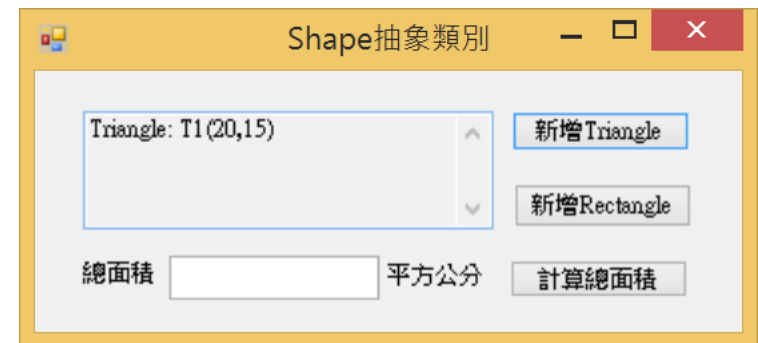
A dialog box titled "新增Triangle" with three text boxes for input: "名稱" (Name) with value "T1", "底" (Base) with value "20", and "高" (Height) with value "15". There are two buttons: "確定" (OK) and "取消" (Cancel). Arrows point from the buttons to text on the right: "設定DialogResult 選取屬性值 OK" for the OK button and "設定DialogResult 選取屬性值 Cancel" for the Cancel button.

設定DialogResult
選取屬性值 OK

設定DialogResult
選取屬性值 Cancel

```
partial class Form2 : Form
```

```
{  
    public void getData(out string name,  
                        out double tbase, out double height)  
    {  
        name = textBox1.Text;  
        tbase = Convert.ToDouble(textBox2.Text);  
        height = Convert.ToDouble(textBox3.Text);  
    }  
}
```

An error message dialog box titled "錯誤訊息" (Error Message) with a red "X" icon. The text inside says "輸入字串格式不正確。" (Input string format is incorrect). There is a "確定" (OK) button at the bottom.A form titled "Shape抽象類別" (Shape Abstract Class). It has a list box containing "Triangle: T1(20,15)". There are two buttons: "新增Triangle" (Add Triangle) and "新增Rectangle" (Add Rectangle). Below the list box, there is a label "總面積" (Total Area) followed by a text box and the unit "平方公分" (square centimeters). There is also a button labeled "計算總面積" (Calculate Total Area).

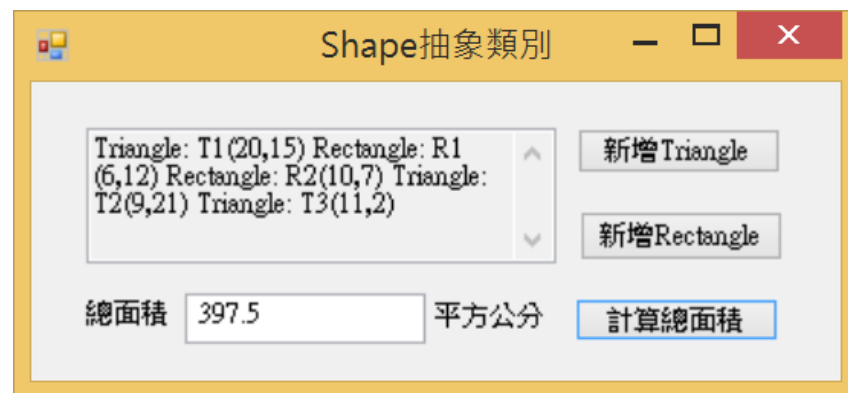
多型 (Polymorphism)

- 一個父類別可以有多個子類別，每個子類別對於父類別的**虛擬方法(virtual)**或**抽象方法(abstract)**可以有不同的**override**方法改寫函式
- **父類別變數**可以**參考不同子類別的物件**，在**runtime**執行時，系統會根據產生的物件是屬於哪個子類別的物件，而執行該子類別的**override**改寫函式

```
List<Shape> allShape = new List<Shape>();
```

- allShape宣告為存放**Shape物件**的串列，可以為子類別的Triangle物件或Rectangle物件
- 新增的物件要到 **runtime** 時才會知道是Triangle或是Rectangle，因此需使用**多型的動態繫結**，根據物件的不同子類別執行不同的show()和area()方法
- 下圖在allShape串列中新增了T1,T2,T3三個Triangle物件及R1,R2兩個Rectangle物件
- 計算總面積時，**allShape[i]**從串列中取出的物件可能為Triangle或是Rectangle，但是多型的**override**會執行適當的area()

```
private void button3_Click(object sender, EventArgs e)
{
    double total = 0.0;
    for (int i = 0; i < allShape.Count; i++)
        total = total + allShape[i].area();
    textBox2.Text = " " + total;
}
```



show()的多型

```
abstract class Shape
```

```
{  
    public virtual string show()  
    {  
        return name;  
    }  
}
```

```
class Triangle : Shape
```

```
{  
    public override string show() {  
        return "Triangle: " + name + "(" + tbase + "," + height + ")";  
    }  
}
```

```
class Rectangle : Shape
```

```
{  
    public override string show()  
    {  
        return "Rectangle: "+name+"("+length+", "+width+")";  
    }  
}
```

```
partial class Form1 : Form  
{
```

```
List<Shape> allShape = new List<Shape>();
```

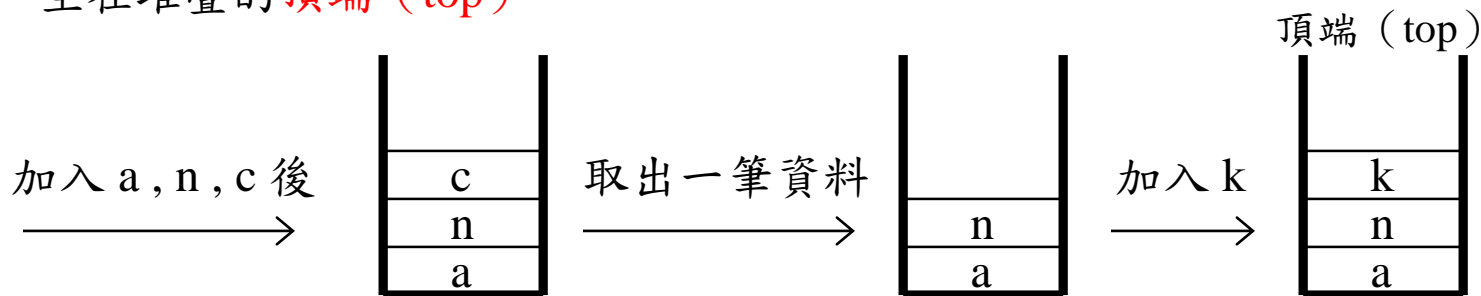
```
public void showShape()  
{
```

```
    int i;  
    textBox1.Text = "";  
    for (i = 0 ; i < allShape.Count ; i++)  
        textBox1.Text = textBox1.Text + allShape[i].show() + " ";  
}
```

堆疊(stack)與佇列(queue)

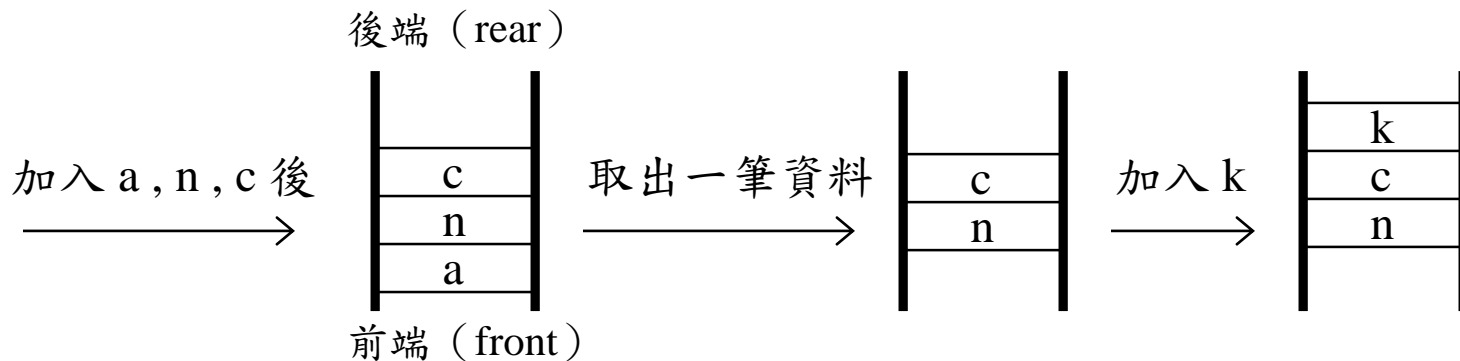
- 堆疊 (stack) : **LIFO** (last in first out 後進先出)

- 堆疊是由一串資料項目所組成的資料結構，資料的**加入**和**取出**都只發生在堆疊的**頂端 (top)**



- 佇列 (queue) : **FIFO** (first in first out 先進先出)

- 佇列是由一串資料項目所組成的資料結構，資料的**加入**發生在佇列的**後端 (rear)**，而資料的**取出**是發生在佇列的**前端 (front)**



堆疊的表示法

- 堆疊的基本運作

- **push**（推入資料）：從頂端**加入**資料，需先判斷堆疊**不是滿的**
- **pop**（取出資料）：從頂端**刪除並傳回**資料，需先判斷堆疊**不是空的**

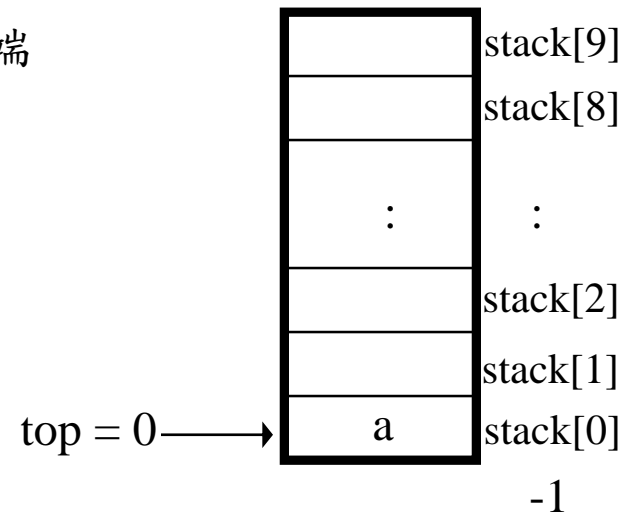
- 使用之資料結構表示法

- 陣列（array）

- 宣告一個一維陣列 `char[] stack= new char[10];`
- 使用一個整數**top**，表示目前堆疊的頂端，top的起始值為 -1
- **top == -1**，表示堆疊為空的
- **top == 9**，表示堆疊為滿的
- push推入資料，則 `top++`
- pop取出資料，則 `top--`

- 鏈結串列（linked list）

- 動態配置，後面再介紹



```
class MyStack
```

```
{
```

```
    const int MAX = 1000;
```

```
    char[] stack= new char[MAX];
```

```
    int top = -1;
```

```
    public void push(char item)
```

```
    {
```

```
        if (top == MAX-1)
```

```
            MessageBox.Show("空間已滿", "錯誤訊息", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
        else
```

```
        {
```

```
            top++;
```

```
            stack[top] = item;
```

```
        }
```

```
    }
```

```
    public bool isEmpty()
```

```
    {
```

```
        if (top < 0)
```

```
            return true;
```

```
        else
```

```
            return false;
```

```
    }
```

```
    public char pop()
```

```
    {
```

```
        char item;
```

```
        item = stack[top];
```

```
        top--;
```

```
        return item;
```

```
    }
```

```
}
```

建立MyStack類別

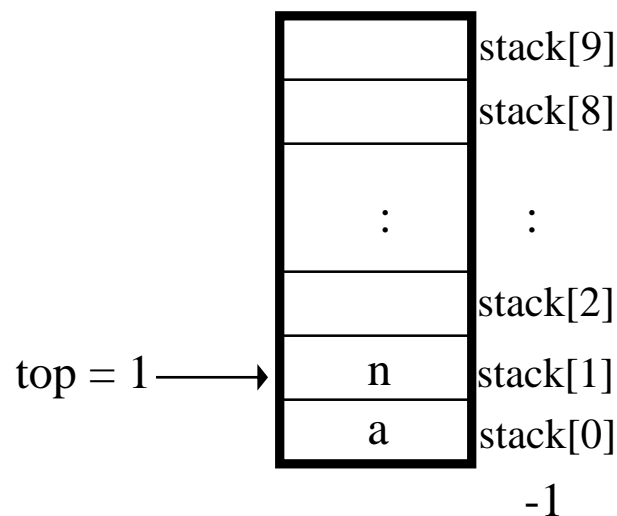
```
char ch;
```

```
MyStack s1 = new MyStack();
```

```
s1.push('a');
```

```
s1.push('n');
```

```
ch = s1.pop();
```



堆疊的應用：運算式的轉換

- 運算式由**運算元**、**運算子**、和**括號**所組成
 - 例如 $a*(b+c)-d$ 為一個運算式，其中 $abcd$ 為運算元， $*+-$ 為運算子
- 運算式求值之**順序**規則
 - 括號優先
 - **由左至右**依運算子的優先順序處理
 - 運算子的優先順序為 乘除 > 加減 > 大於,等於,小於 > and > or
- 運算式的表示法有三種
 - **中序式 (infix)**：運算子放在運算元中間，例如 $a + b$
 - **前序式 (prefix)**：運算子放在運算元前面，例如 $+ a b$
 - **後序式 (postfix)**：運算子放在運算元後面，例如 $a b +$
- 中序式有括號，但前序式與後序式則無括號
 - 中序式最適合讓人來讀取運算式，但不適合電腦來讀取，因為還要處理括號
 - 通常電腦要計算一個運算式之值時，需先將中序式轉換為後序式

中序式轉換為前序式和後序式

- 按照中序式求值之優先順序，將運算子置於兩個運算元的前面（前序式）或後面（後序式）
 - 例1：中序式 $A - B * (C + D) / E$ 可以轉換為
 - 前序式 $- A / * B + C D E$
 - 後序式 $A B C D + * E / -$
 - 例2：中序式 $B - C + D * E / (F - G)$ 可以轉換為
 - 前序式 $+ - B C / * D E - F G$
 - 後序式 $B C - D E * F G - / +$

使用堆疊將中序式轉換為後序式

- 由左至右掃描中序運算式，遇到
 - 運算元：直接輸出到後序式中
 - 左括號'('：push() 推入堆疊中
 - 右括號')'：重覆pop()取出堆疊中運算子到後序式中，直到取出一個左括號為止
 - 運算子：當目前運算子的優先權 \leq 位於頂端(top)運算子的優先權時，重覆pop()取出頂端的運算子到後序式中，直到目前運算子的優先權較大為止(或頂端非運算子為止)，最後將目前運算子push() 推入堆疊中
- 掃描完中序式後，將堆疊中的所有運算子pop()取出到後序式中，直到清空堆疊為止

| Token | a | * | (| b | + | c |) | / | d | clear |
|----------|----|---|---|----|----|-----|------|-------|--------|---------|
| Stack[2] | | | | | + | + | | | | |
| Stack[1] | | | (| (| (| (| | | | |
| Stack[0] | | * | * | * | * | * | * | / | / | |
| Top | -1 | 0 | 1 | 1 | 2 | 2 | 0 | 0 | 0 | -1 |
| Output | a | a | a | ab | ab | abc | abc+ | abc+* | abc+*d | abc+*d/ |

佇列(Queue)的表示法

- 佇列的基本運作

- **Enqueue** (加入資料)：從**後端**加入資料，需先判斷佇列**不是滿的**
- **Dequeue** (取出資料)：從**前端**刪除並傳回資料，需先判斷佇列**不是空的**

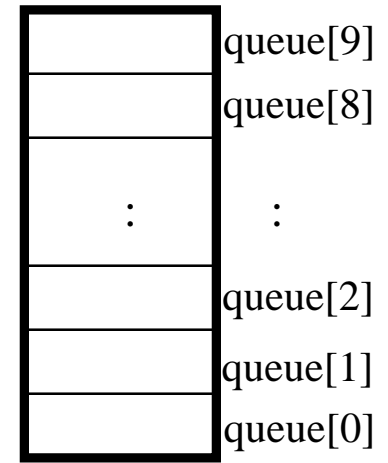
- 使用之資料結構表示法

- 陣列 (array)

- 宣告一個一維陣列 `int[] queue = new int[10];`
- 使用兩個整數 `front` 和 `rear`，**front**為目前佇列前端的前一個編號(前端減1)，**rear**為目前佇列的後端，起始值 `front = -1`，`rear = -1`
- **front == rear**，表示佇列為空的
- **rear == 9**，表示佇列為滿的
- 加入資料，則 `rear++`
- 取出資料，則 `front++`

- 鏈結串列 (linked list)

- 動態配置，後面再介紹



`rear = -1` \longrightarrow
`front = -1` \longrightarrow

```
class MyQueue
```

```
{
```

```
    const int MAX = 1000;
```

```
    int[] queue= new int[MAX];
```

```
    int front = -1, rear=-1;
```

```
    public void enqueue(int item)
```

```
    {
```

```
        if (rear == MAX-1)
```

```
            MessageBox.Show("空間已滿", "錯誤訊息", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
        else
```

```
        {
```

```
            rear++;
```

```
            queue[rear] = item;
```

```
        }
```

```
    }
```

```
    public bool isEmpty()
```

```
    {
```

```
        if (front == rear)
```

```
            return true;
```

```
        else
```

```
            return false;
```

```
    }
```

```
    public int dequeue()
```

```
    {
```

```
        front++;
```

```
        return queue[front];
```

```
    }
```

```
}
```

建立MyQueue類別

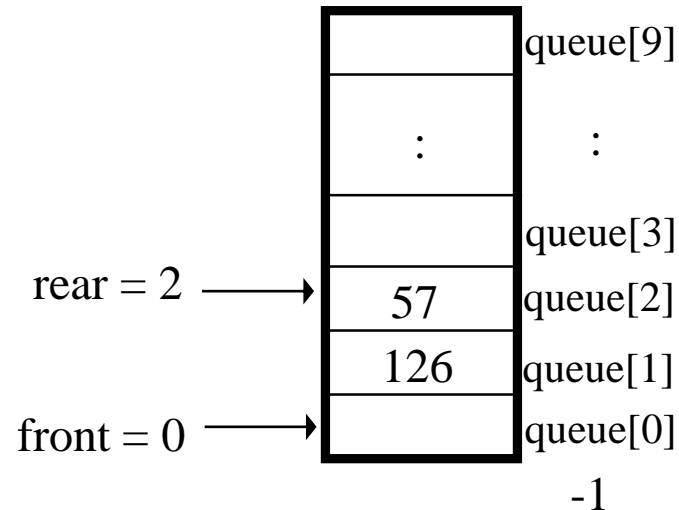
```
MyQueue q1 = new MyQueue();
```

```
q1.enqueue(78);
```

```
q1.enqueue('126');
```

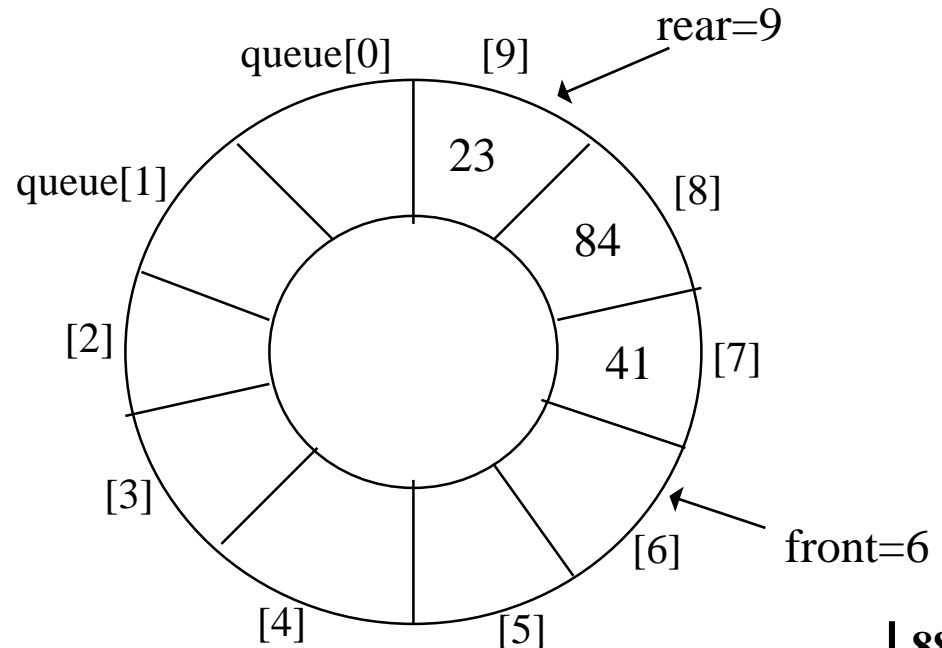
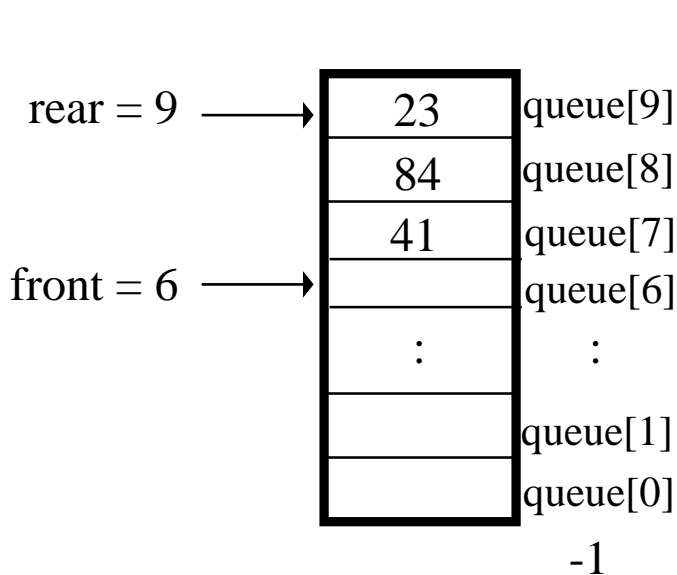
```
int n = q1.dequeue();
```

```
q1.enqueue(57);
```



環狀佇列 (Circular Queues)

- 前面所介紹的佇列表示法和演算法，會發生浪費空間的現象，如下左圖所示，雖然還有空間，但因 **rear==9 佇列已滿**，無法再加入資料
- 為了解決浪費空間的問題，佇列常常以如下右圖**環狀**的方式來運作
 - 加入資料： $\text{rear} = (\text{rear} + 1) \% 10$;
 - 取出資料： $\text{front} = (\text{front} + 1) \% 10$;
- 若連續enqueue七次， $(\text{rear}+7)\%10=6$ ， $\text{rear}==\text{front}$ 反而判斷成為空佇列，因此**佇列已滿**的條件改為 **$\text{front}==(\text{rear}+1)\%10$** ，最多只能存9個資料



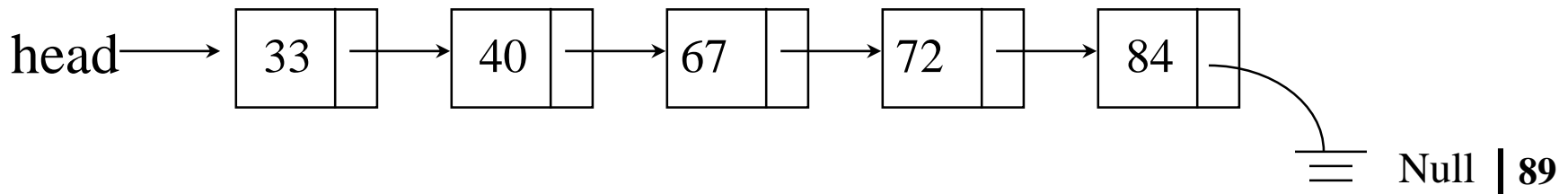
鏈結串列 (Linked List)

- 鏈結串列 VS 陣列

- 陣列：靜態串列，陣列宣告之後元素個數（陣列長度）就固定
 - 每個資料項儲存在一個陣列元素中，按照陣列註標的順序排列

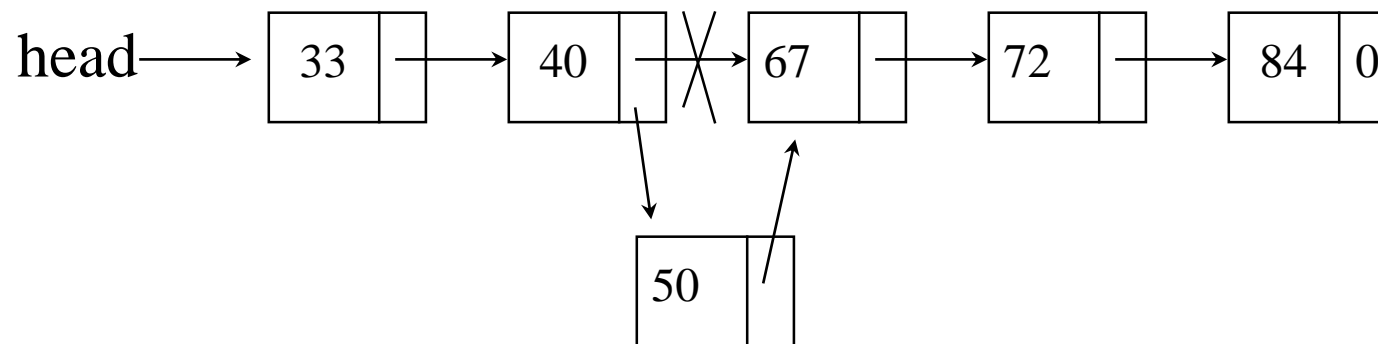
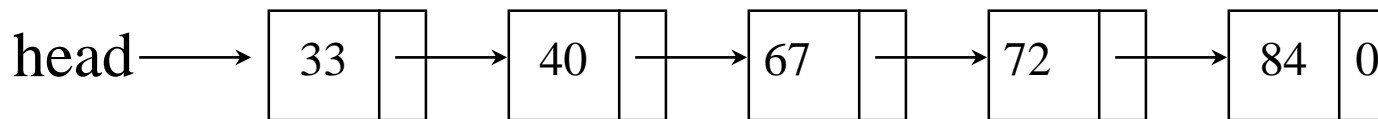
| n[0] | n[1] | n[2] | n[3] | n[4] | n[5] | n[6] | n[7] | n[8] | n[9] |
|------|------|------|------|------|------|------|------|------|------|
| 33 | 40 | 67 | 72 | 84 | | | | | |

- 鏈結串列：動態串列，元素個數可以動態改變
 - 由許多節點所組成，每個節點包含兩個欄位：資料欄和鏈結欄
 - 每個資料項儲存在節點中的資料欄中
 - 鏈結欄為一個指標，指向此節點的下一個節點位址



加入 50：（假設所有資料需排序，照大小順序存放）

Null



n[0] n[1] n[2] n[3] n[4] n[5] n[6] n[7] n[8] n[9]

| | | | | | | | | | |
|----|----|----|----|----|--|--|--|--|--|
| 33 | 40 | 67 | 72 | 84 | | | | | |
|----|----|----|----|----|--|--|--|--|--|

| | | | | | | | | | |
|----|----|--|----|----|----|--|--|--|--|
| 33 | 40 | | 67 | 72 | 84 | | | | |
|----|----|--|----|----|----|--|--|--|--|

先往右移動

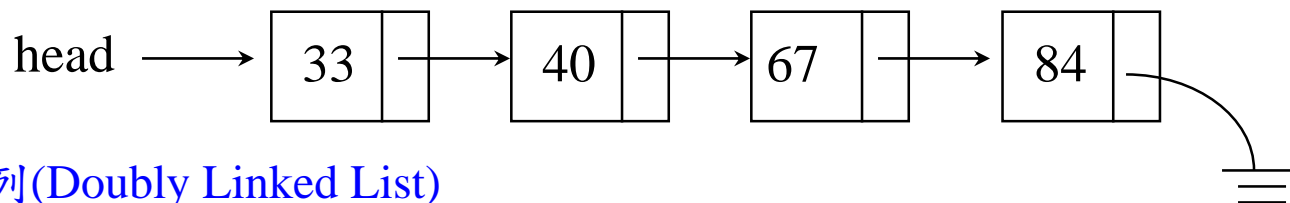
| | | | | | | | | | |
|----|----|----|----|----|----|--|--|--|--|
| 33 | 40 | 50 | 67 | 72 | 84 | | | | |
|----|----|----|----|----|----|--|--|--|--|

再加入50

單向鏈結串列與雙向鏈結串列

- 單向鏈結串列(Singly Linked List)

- 每個節點只有一個鏈結欄，指向下一個節點（單一方向）



- 雙向鏈結串列(Doubly Linked List)

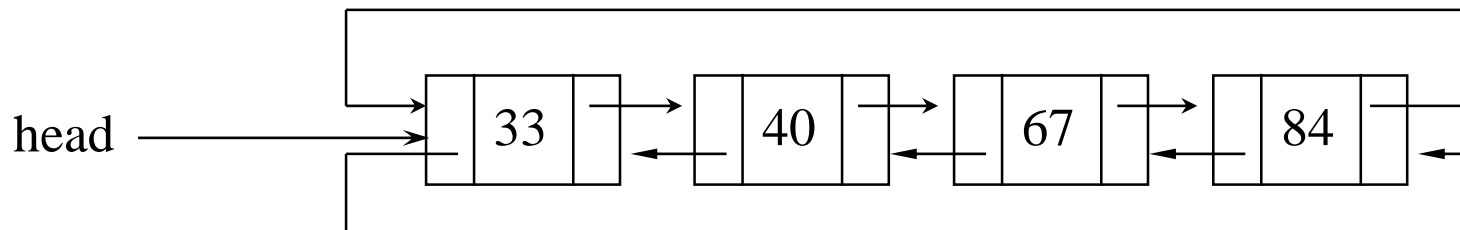
- 每個節點有兩個鏈結欄，分別指向前一個節點和後一個節點（兩個方向）

- 優點

- 當任何一個鏈結斷裂時，容易藉由反方向的鏈結，來復原已斷裂的鏈結
- 搜尋節點的工作會比較容易和快速，因為任何一個節點都可以藉由雙方向的鏈結，很容易地找到前一個及後一個節點

- 缺點

- 每個節點都增加了一個鏈結欄（Llink），比較浪費空間
- 加入與刪除節點的工作負擔是單向鏈結串列的兩倍（加入新節點時，雙向需改變四個鏈結指標，而單向需改變兩個。刪除節點時，雙向需改變兩個鏈結指標，而單向需改變一個）

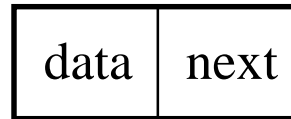


單向鏈結串列(Singly Linked Lists)

- 單向鏈結串列的資料結構表示法 (C語言 VS. C#語言)
 - 由節點(node)串接而成，每個節點包含資料欄和鏈結欄
 - C語言利用struct定義節點結構，C#語言改用類別定義節點

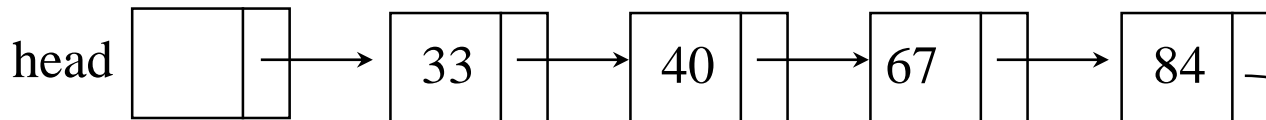
```
struct node
{
    int data;
    struct node *next;
};
```

node



```
class node
{
    int data;
    node next;
    public node(int n)
    {
        data = n;
        next = null;
    }
}
```

- 產生節點：C語言利用指標指向動態配置空間，C#語言改用new一個物件
 - (C語言) `struct node *x = (struct node*) malloc(sizeof(struct node));`
`x->data = n; x->next = NULL;`
 - (C#語言) `node x = new node(n);`
- C語言利用一個節點指標 head，指向鏈結串列的第一個節點，以便由此為起點開始尋找資料，C#語言改用一個head物件為起點(但head物件的data欄值無意義)



單向鏈結串列的加入資料 (Insert)

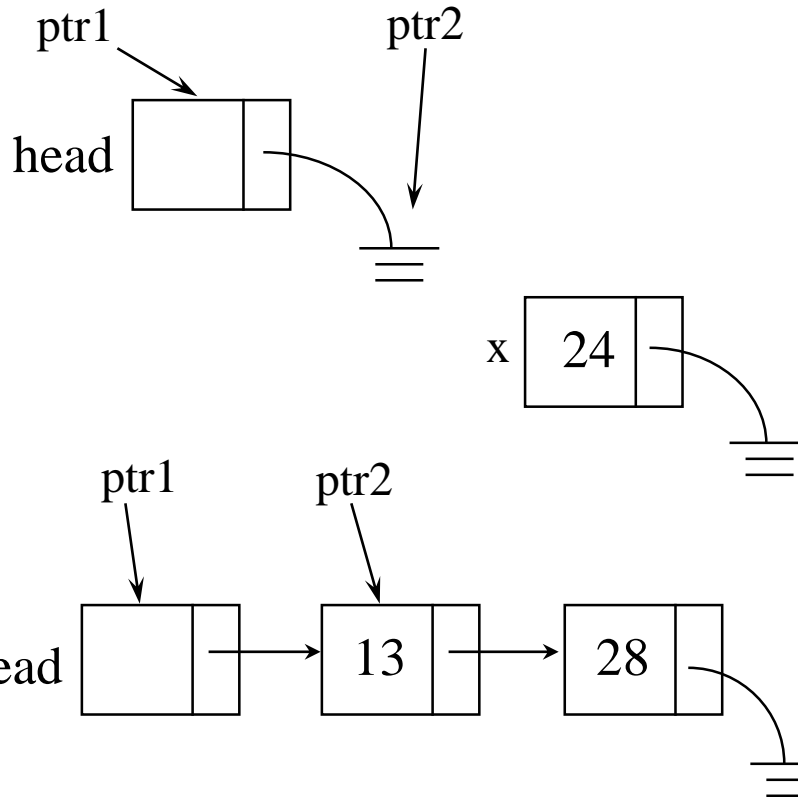
- 有序鏈結串列(由小到大)中加入資料節點 (假設加入整數 n)

- 產生一個新節點 x，資料欄放入整數 n

- node x = new node(n);

- 找到兩個節點 ptr1 和 ptr2，其中 ptr1 的 data 必須小於等於 n，ptr2 的 data 則大於 n，中間可以加入 n

```
node x = new node(n);
node ptr1 = head;
node ptr2 = head.getNext();
while (ptr2 != null)
{
    if (ptr2.getData() > n)
    {
        ptr1.setNext(x);
        x.setNext(ptr2);
        return;
    }
    ptr1 = ptr2;
    ptr2 = ptr2.getNext();
}
ptr1.setNext(x);
```

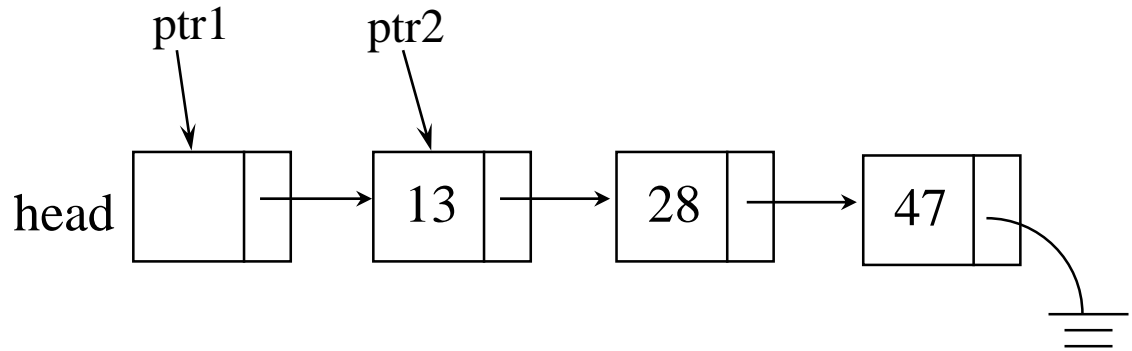


```
class node
{
    int data;
    node next;
    public node(int n)
    {
        data = n;
        next = null;
    }
    public int getData()
    {
        return data;
    }
    public node getNext()
    {
        return next;
    }
    public void setData(int n)
    {
        data = n;
    }
    public void setNext(node d)
    {
        next = d;
    }
}
```

單向鏈結串列的刪除資料 (Delete)

- 在有序鏈結串列中刪除資料項節點 (假設刪除整數 n)
 - 從 head 開始往下找節點，找到所需刪除的節點 ptr2 (資料欄為整數 n)，前一個節點為 ptr1，將 ptr 1 的鏈結欄指向 ptr2 的下一個節點即可

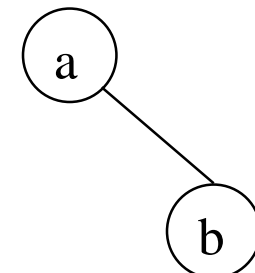
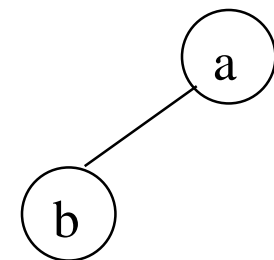
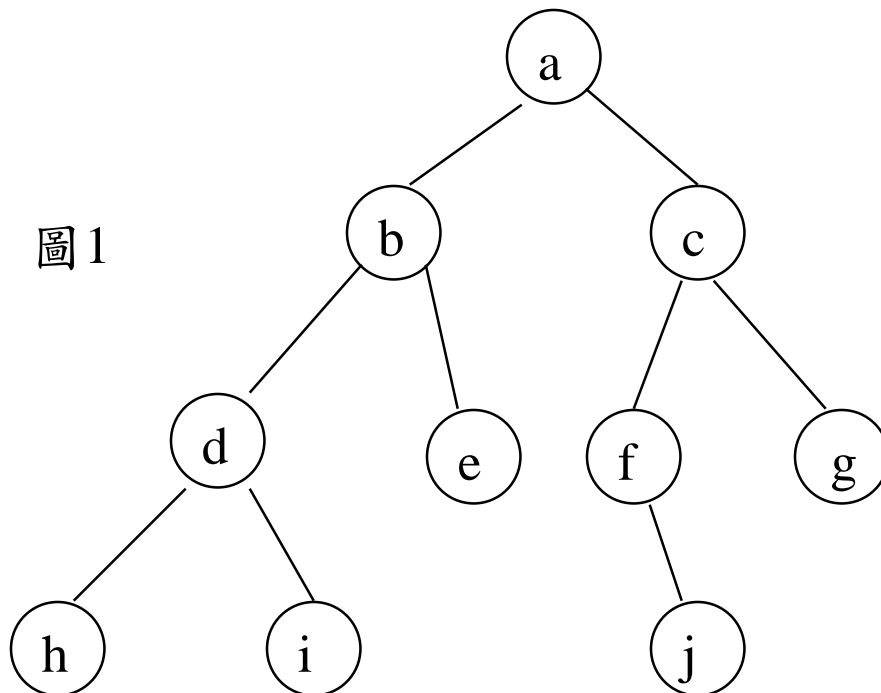
```
node ptr1 = head;
node ptr2 = head.getNext();
while (ptr2 != null)
{
    if (ptr2.getData() == n)
    {
        ptr1.setNext(ptr2.getNext());
        return;
    }
    if (ptr2.getData() > n)
        throw new Exception("串列中沒有"+n+ "，無法刪除");
    ptr1 = ptr2;
    ptr2 = ptr2.getNext();
}
throw new Exception("串列中沒有"+n+ "，無法刪除");
```



樹狀結構：二元樹(Binary Tree)

- 二元樹 (binary tree)

- 每個節點的子節點個數 (分支度 degree) **最多兩個**，一般樹則無此限制
 - 左子節點(left child)，右子節點(right child)
- 二元樹節點之間有**左右順序關係** (圖2不等於圖3)，一般樹則無順序關係

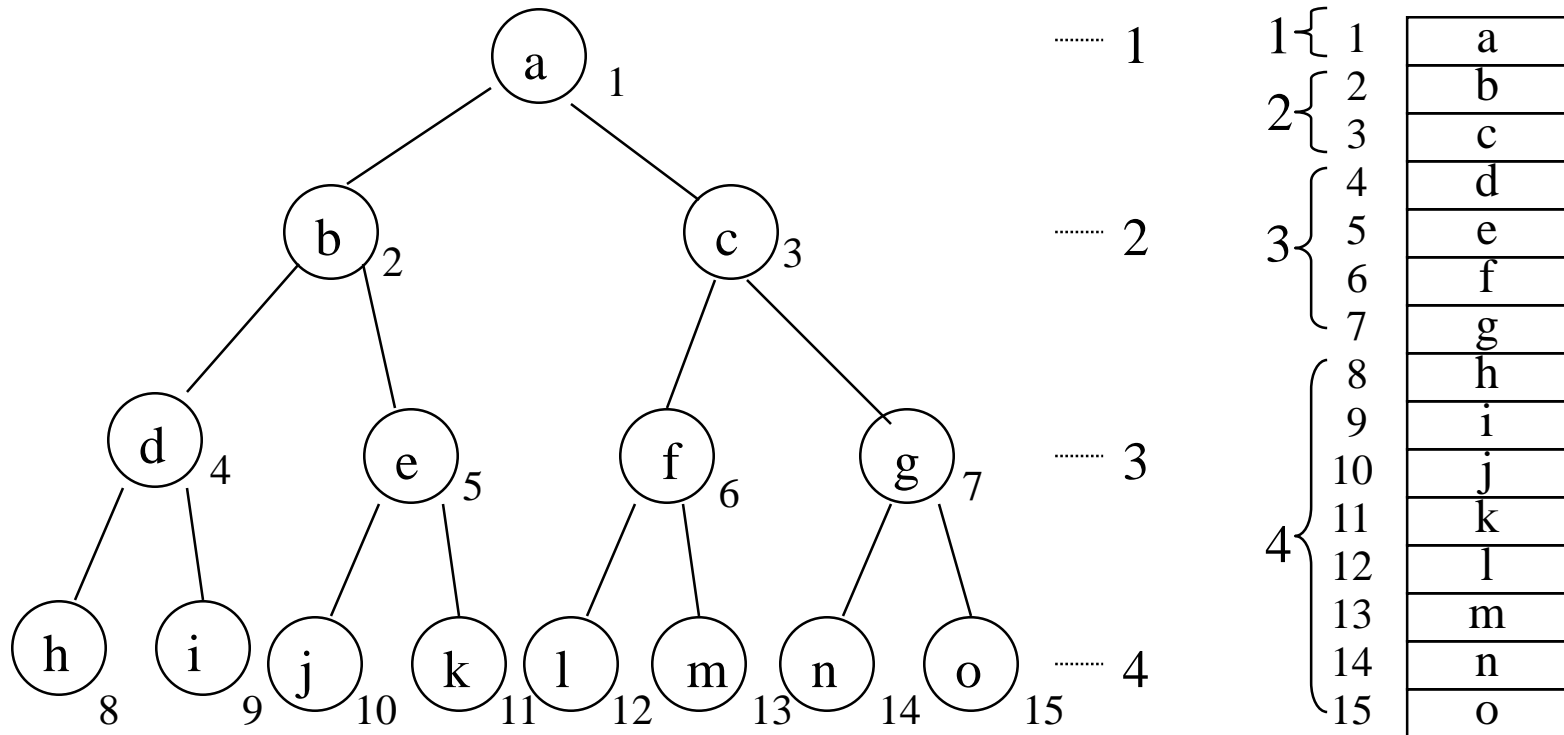


二元樹的陣列表示法

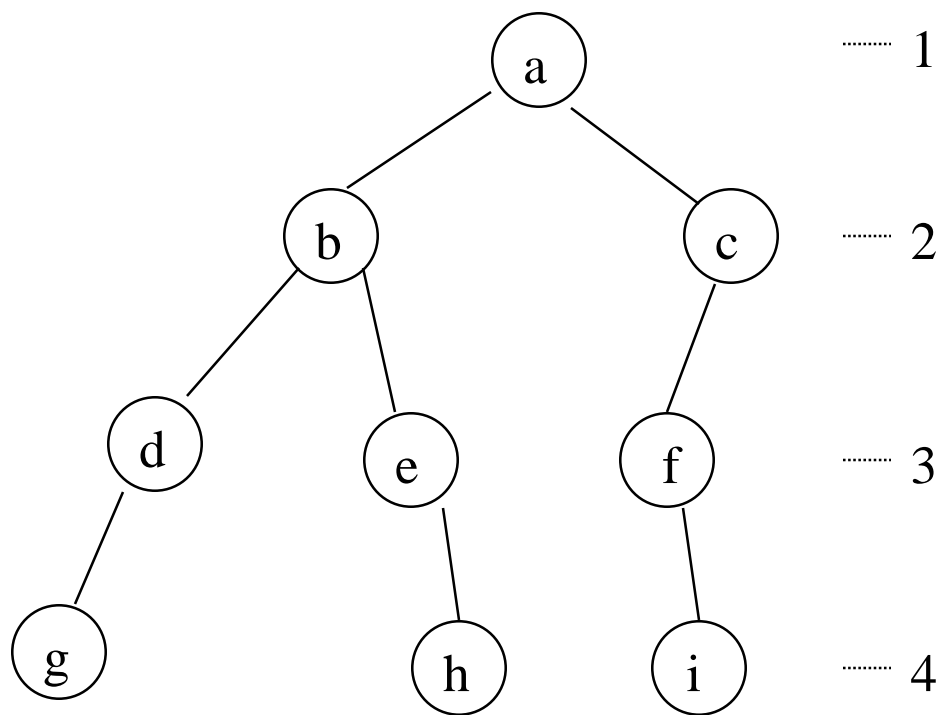
一維陣列表示法

– 對於一個高度為 n 的二元樹，可以用一個長度為 $2^n - 1$ 的陣列來表示

- 因為高度為 n 的二元樹最多只有 $2^n - 1$ 個節點（full binary tree）



例子：二元樹之一維陣列表示法



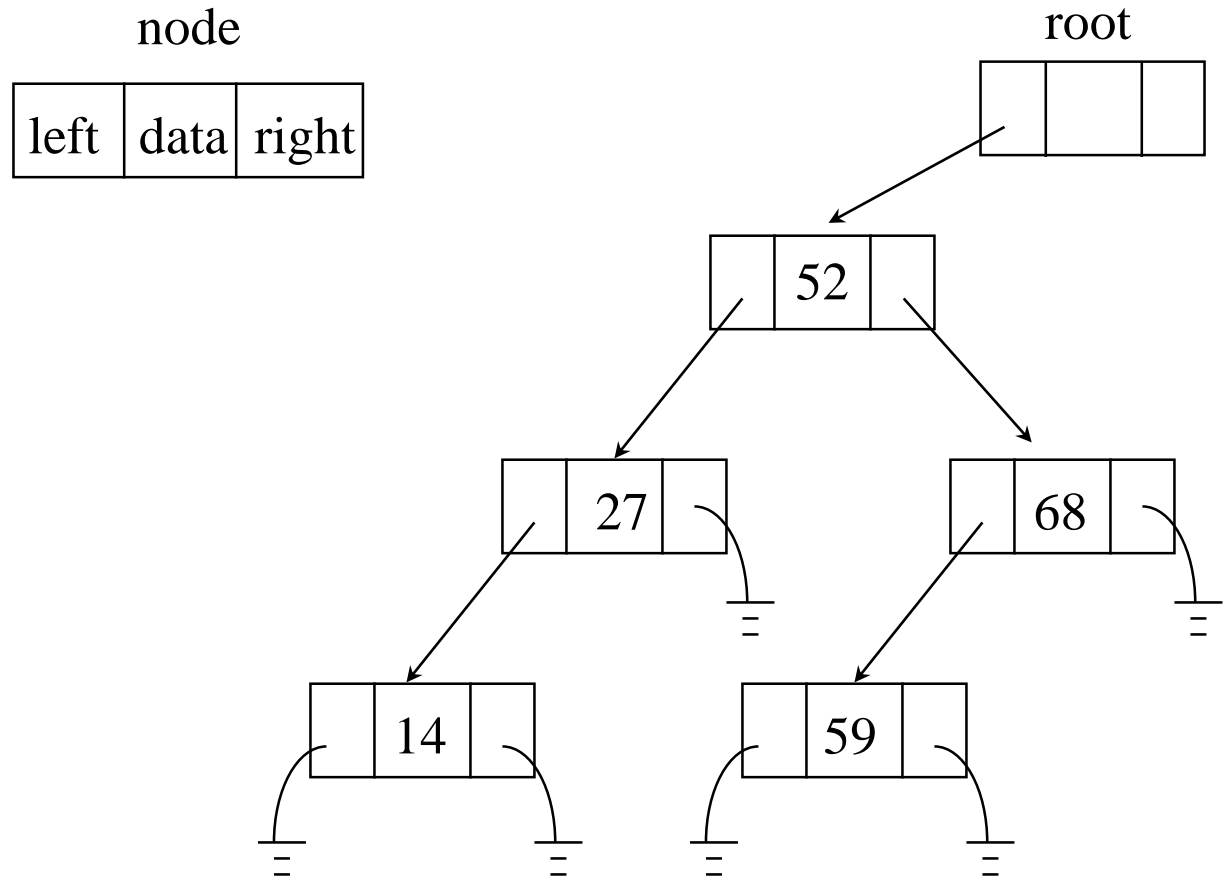
| | |
|----|---|
| 1 | a |
| 2 | b |
| 3 | c |
| 4 | d |
| 5 | e |
| 6 | f |
| 7 | |
| 8 | g |
| 9 | |
| 10 | |
| 11 | h |
| 12 | |
| 13 | i |
| 14 | |
| 15 | |

```
node root = new node(0);
```

```
class node
```

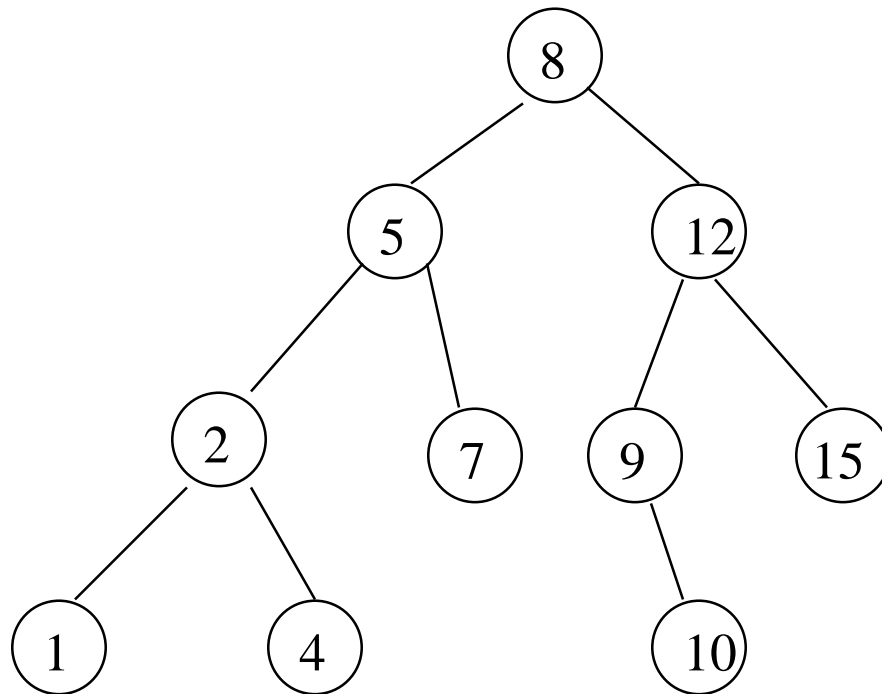
```
{  
    int data;  
    node left;  
    node right;  
    public node(int n)  
    {  
        data = n;  
        left = null;  
        right = null;  
    }  
    public int getData()  
    { return data; }  
    public node getLeft()  
    { return left; }  
    public node getRight()  
    { return right; }  
    public void setData(int n)  
    { data = n; }  
    public void setLeft(node d)  
    { left = d; }  
    public void setRight(node d)  
    { right = d; }  
}
```

二元樹的鏈結串列表示法



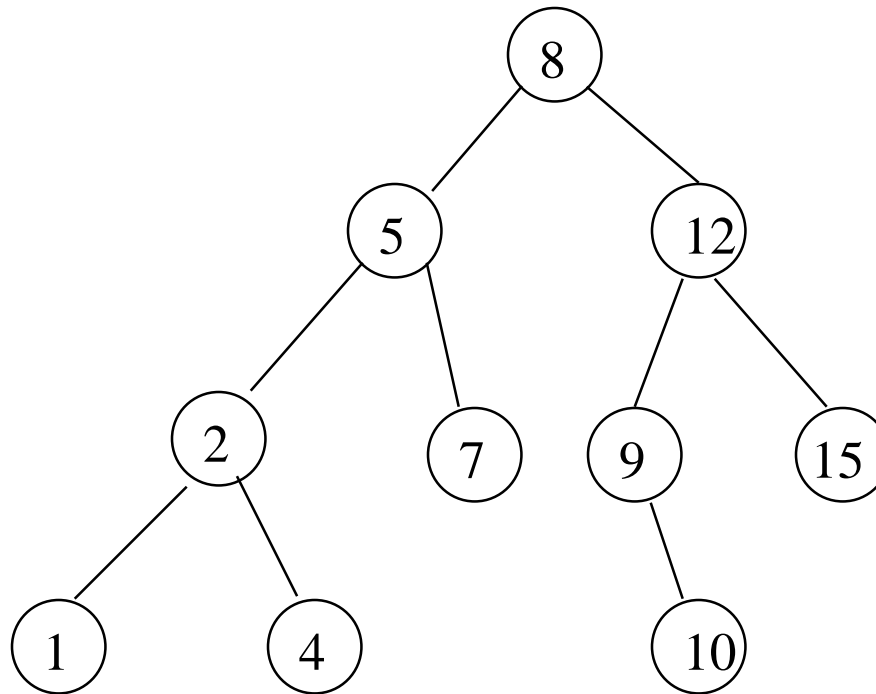
二元搜尋樹(Binary Search Tree)

- 二元搜尋樹 (binary search tree)
 - 二元樹的一種，所有節點的資料值都是唯一的(不可重複)
 - 資料值：左子樹 < 根節點 < 右子樹 (左 < 中 < 右)



- 樹狀結構可以用來儲存資料，由於二元搜尋樹為有序樹（左<中<右），所以不同的資料輸入順序會產生不同的二元樹

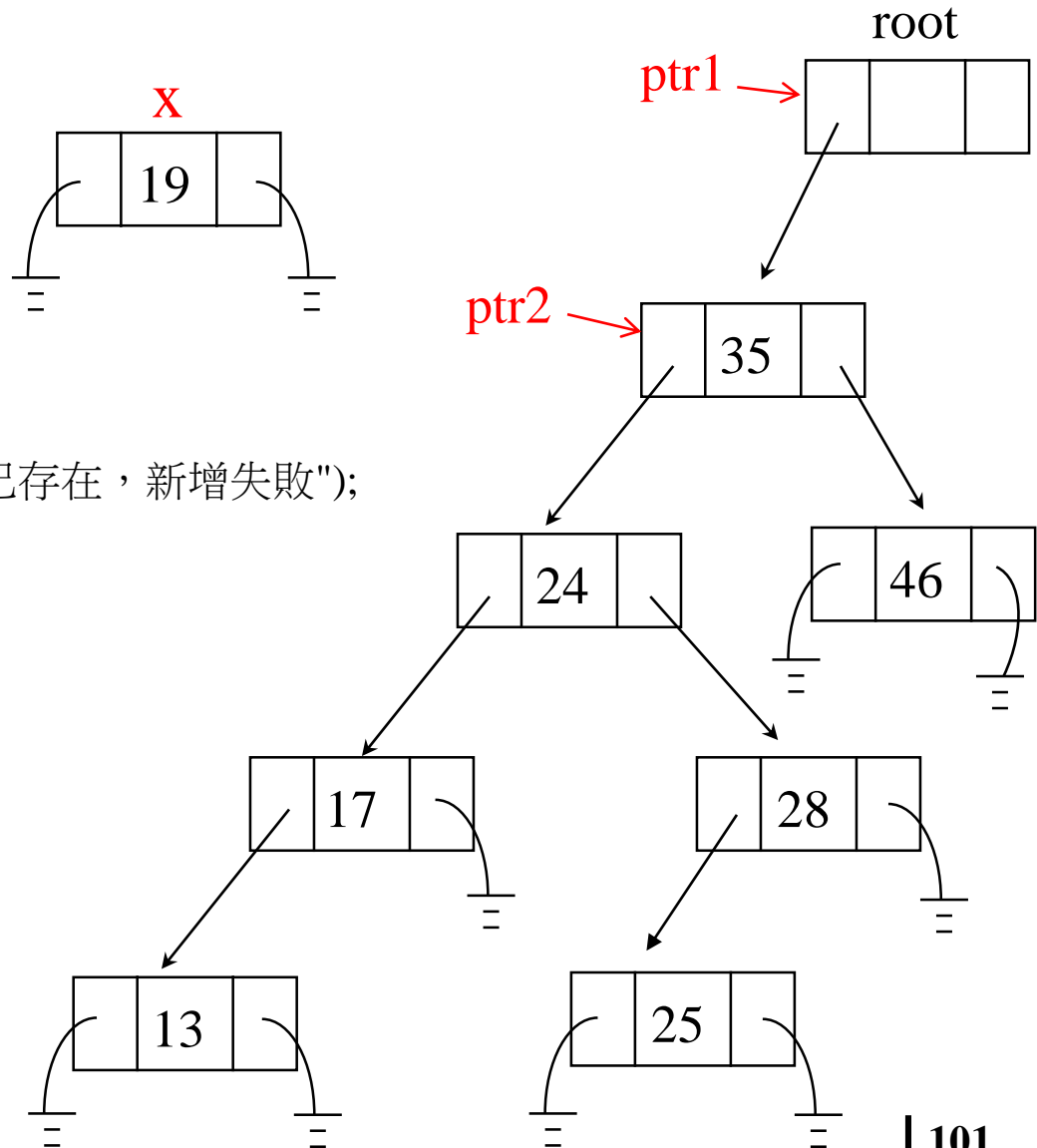
1.輸入順序：8, 12, 5, 9, 10, 2, 1, 7, 15, 4



2.輸入順序：10, 15, 8, 4, 12, 1, 2, 7, 5, 9

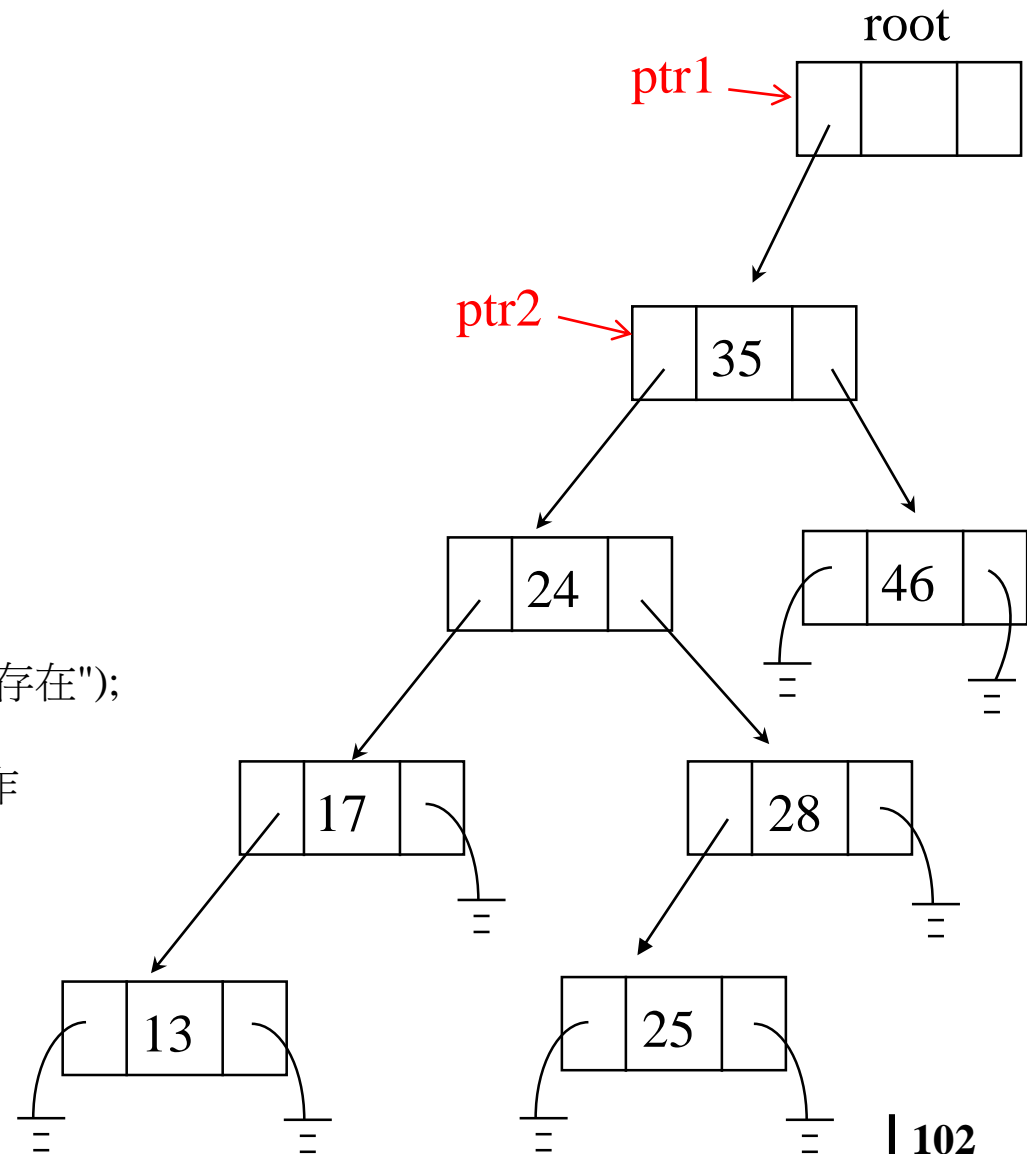
二元搜尋樹的加入資料

```
node ptr1 = root;  
node ptr2 = root.getLeft();  
node x = new node(n);    //加入資料n  
if (ptr2 == null)  
    { root.setLeft(x); return; }  
while (ptr2 != null)  
{  
    if (ptr2.getData() == n)  
        throw new Exception ("資料" + n + "已存在，新增失敗");  
    ptr1 = ptr2;  
    if (n < ptr2.getData())  
        ptr2 = ptr2.getLeft();  
    else  
        ptr2 = ptr2.getRight();  
}  
if (n < ptr1.getData())  
    ptr1.setLeft(x);  
else  
    ptr1.setRight(x);
```



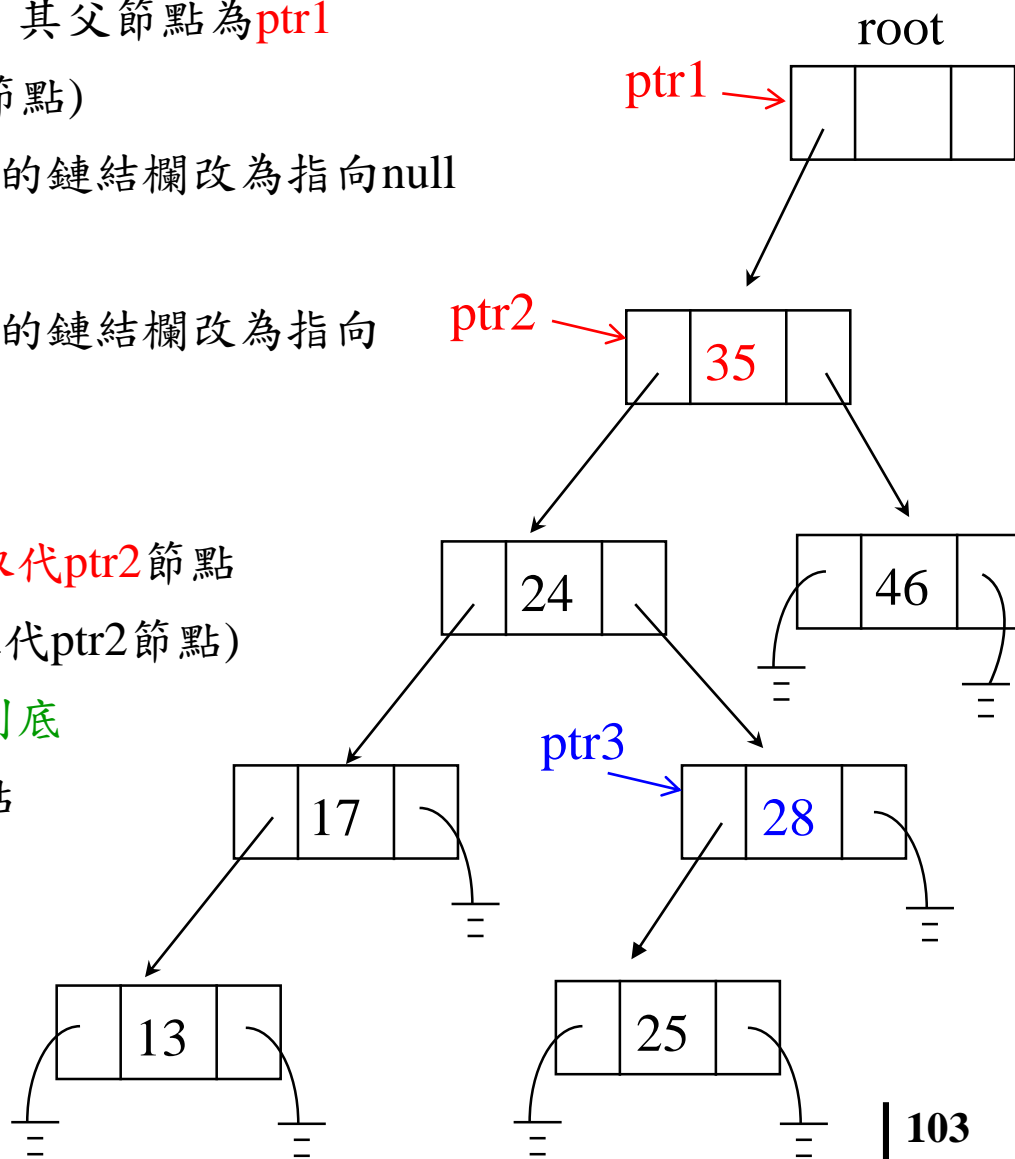
二元搜尋樹的搜尋資料

```
node ptr1 = root;  
node ptr2 = root.getLeft();  
while (ptr2 != null)  
{  
    if (ptr2.getData() == n) //搜尋資料n  
        break;  
    ptr1 = ptr2;  
    if (n < ptr2.getData())  
        ptr2 = ptr2.getLeft();  
    else  
        ptr2 = ptr2.getRight();  
}  
if (ptr2 == null)  
    throw new Exception ("資料" + n + "不存在");  
else  
    ..... // 搜尋到資料n在ptr2，執行工作
```



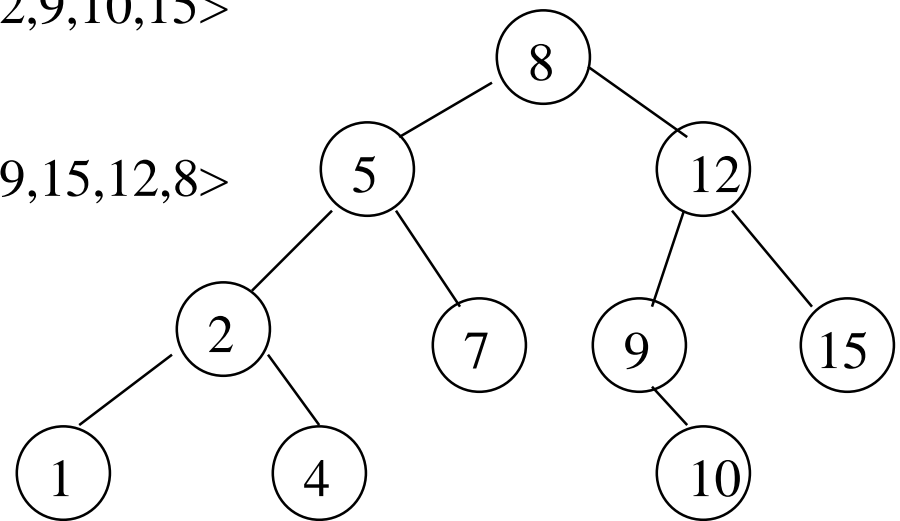
二元搜尋樹的刪除資料

- 先搜尋欲刪除資料n所在的節點ptr2，其父節點為ptr1
- 若要刪除的ptr2節點沒有子節點（葉節點）
 - 直接將父節點ptr1指向此ptr2節點的鏈結欄改為指向null
- 若要刪除的ptr2節點只有一個子節點
 - 直接將父節點ptr1指向此ptr2節點的鏈結欄改為指向ptr2節點的唯一子節點
- 若要刪除的ptr2節點有兩個子節點
 - 左子樹中資料值最大的節點ptr3取代ptr2節點（或是右子樹中資料值最小的節點取代ptr2節點）
 - ptr2節點的左兒子，一直往右走到底即為左子樹中資料值最大的ptr3節點
 - 刪除ptr3節點會造成遞迴



二元樹的走訪方式 (Traversal)

- 二元樹中的每個節點都必須經過一次
- 三種走訪方式
 - 中序走訪(inorder traversal)
 - 左→中→右 <1,2,4,5,7,8,9,10,12,15>
 - 前序走訪(preorder traversal)
 - 中→左→右 <8,5,2,1,4,7,12,9,10,15>
 - 後序走訪(postorder traversal)
 - 左→右→中 <1,4,2,7,5,10,9,15,12,8>



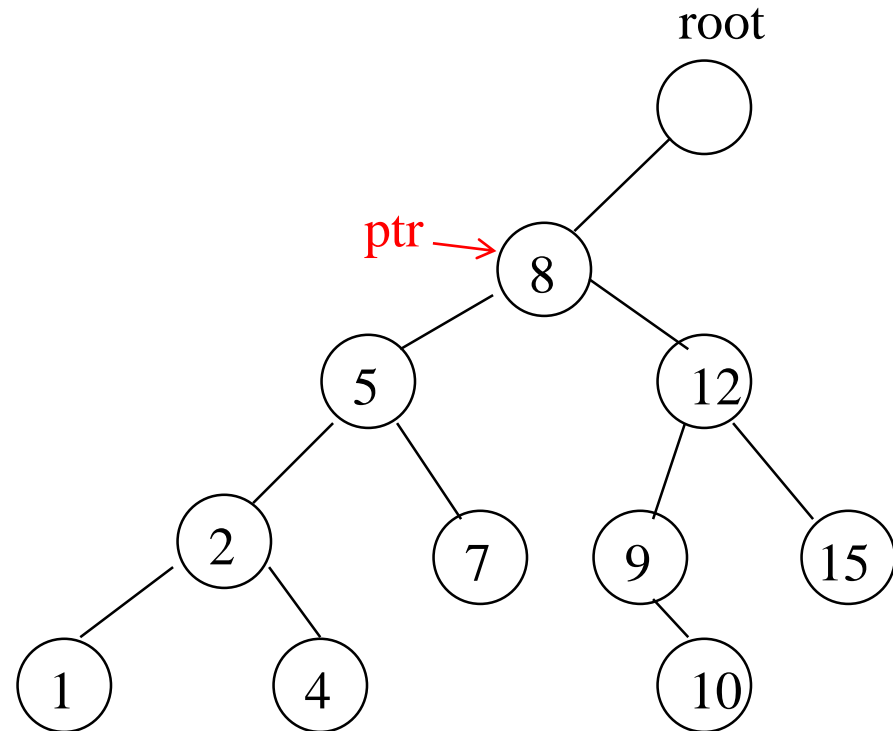
二元樹走訪的演算法

```
void inorder(node ptr)
{
    if (ptr != null)
    {
        inorder(ptr.getLeft());
        textBox2.Text = textBox2.Text + " " + ptr.getData();
        inorder(ptr.getRight());
    }
}
```

```
void preorder(node ptr)
{
    if (ptr != null)
    {
        textBox2.Text = textBox2.Text + " " + ptr.getData();
        preorder(ptr.getLeft());
        preorder(ptr.getRight());
    }
}
```

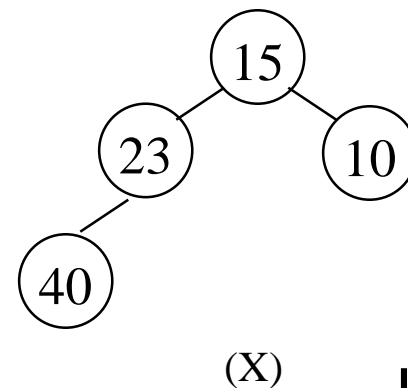
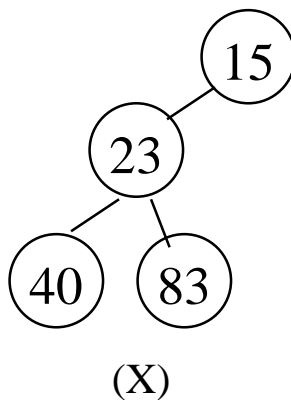
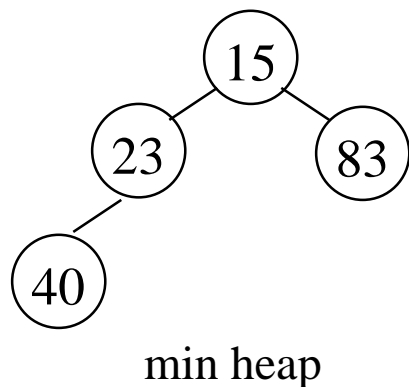
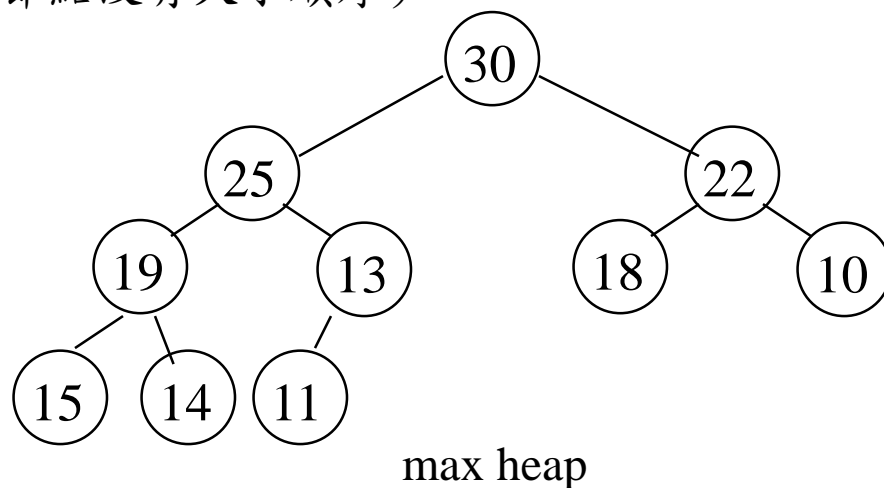
```
void postorder(node ptr)
{
    if (ptr != null)
    {
        postorder(ptr.getLeft());
        postorder(ptr.getRight());
        textBox2.Text = textBox2.Text + " " + ptr.getData();
    }
}
```

```
textBox2.Text="";
inorder(root.getLeft());
preorder(root.getLeft());
postorder(root.getLeft());
```



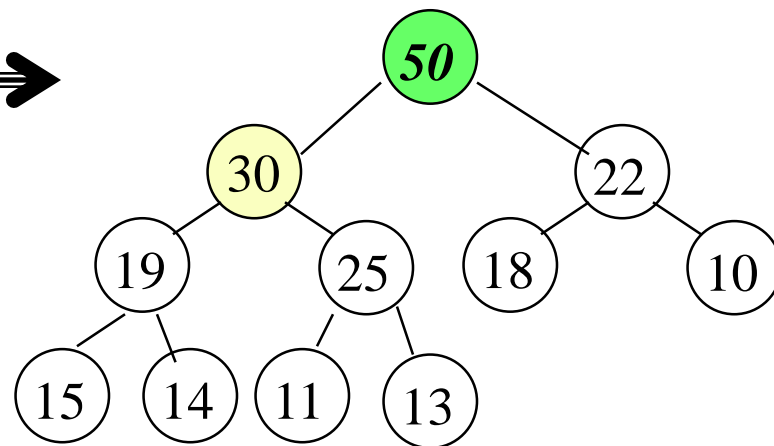
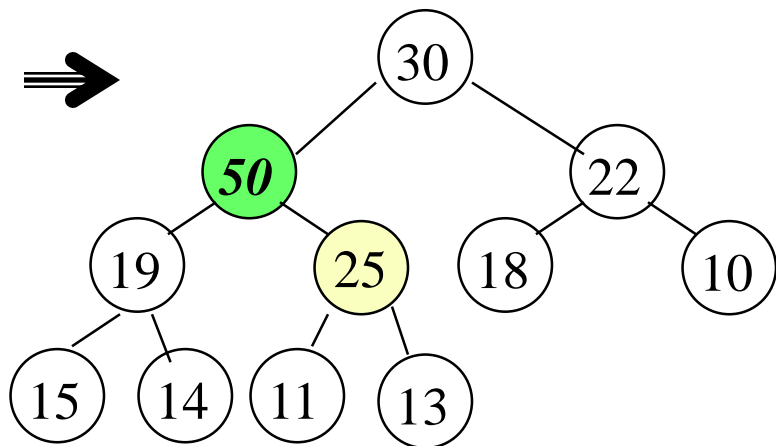
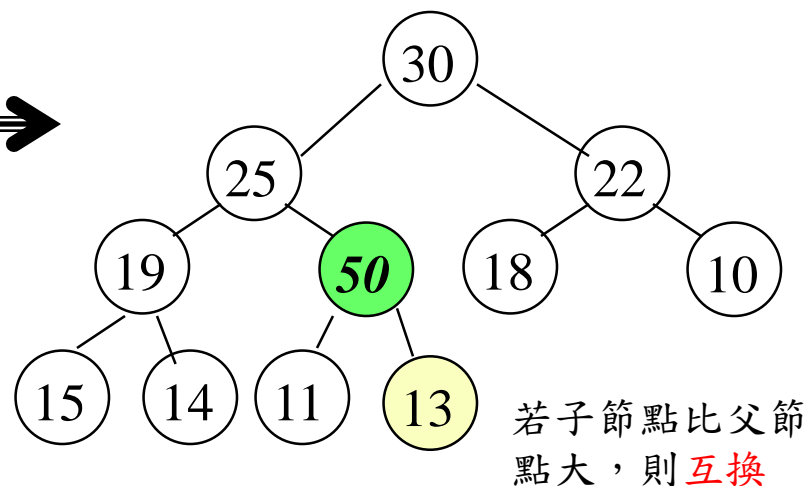
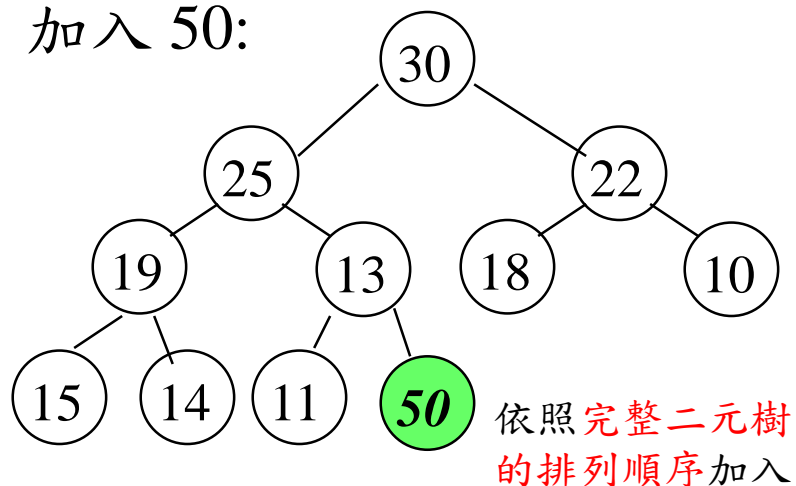
堆積(Heap)

- 完整二元樹(complete binary tree)的一種(但非二元搜尋樹)
 - 最大堆積樹(max heap)
 - 根節點 \geq 子樹節點 (左右子節點沒有大小順序)
 - 最小堆積樹(min heap)
 - 根節點 \leq 子樹節點
- 常用來儲存動態配置記憶體의變數，
有序性的存取速度($O(\log_2 n)$)
比鏈結串列($O(n)$)快
 - 堆積排序法(heap sort)

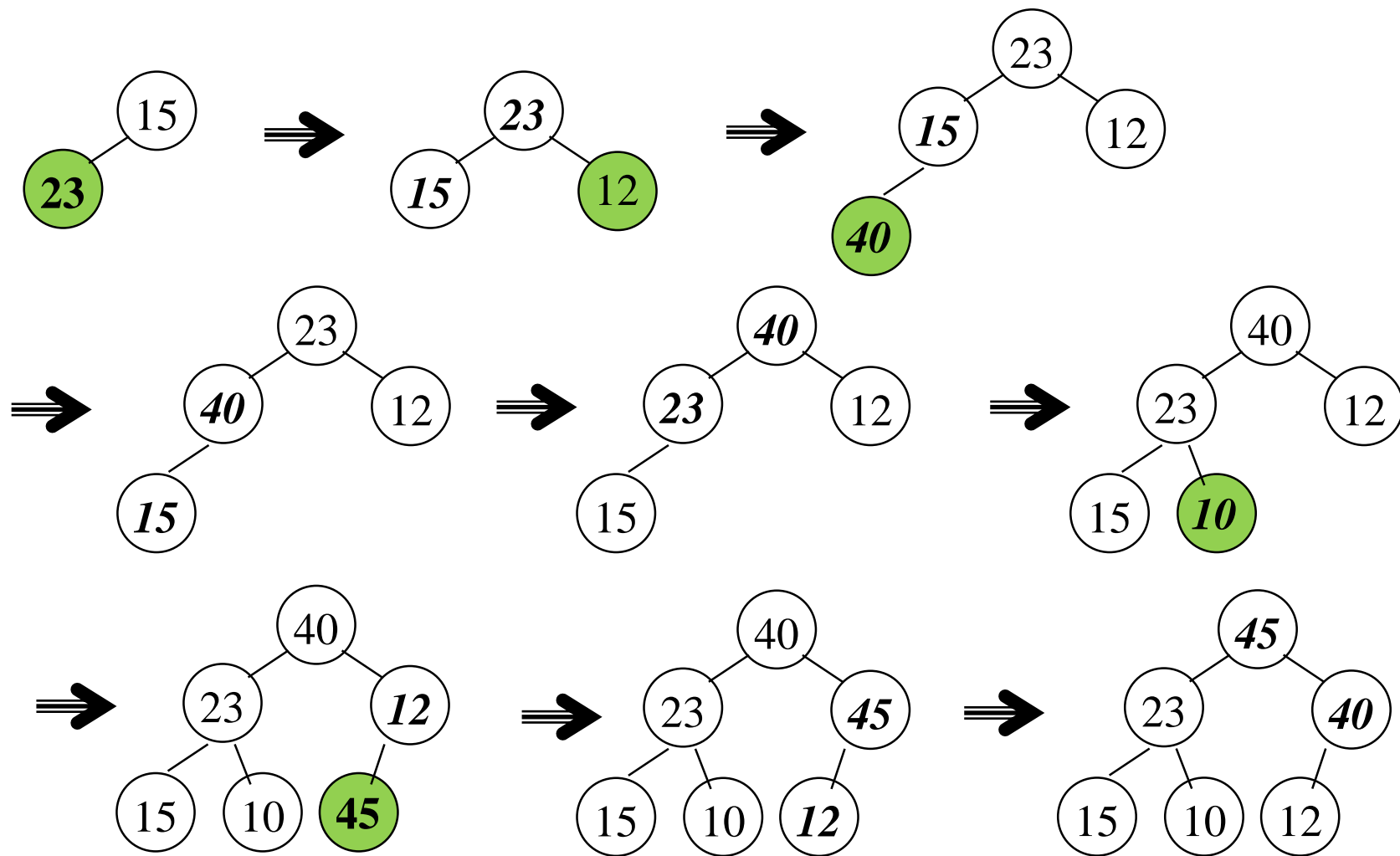


最大堆積樹(max heap)：加入節點

- 加入 50:

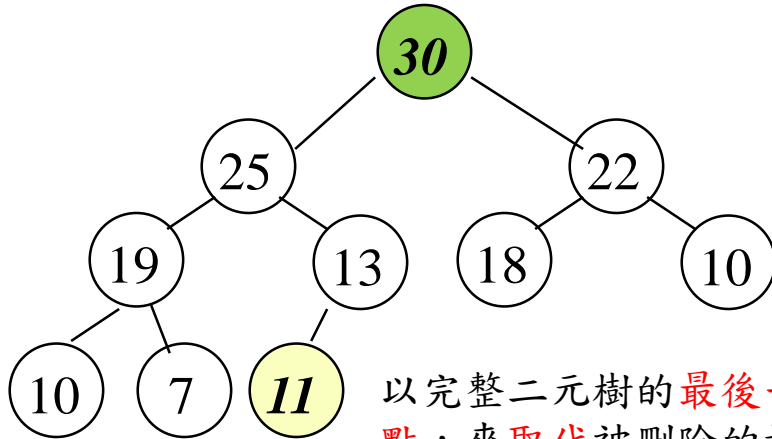


- 輸入順序：15, 23, 12, 40, 10, 45

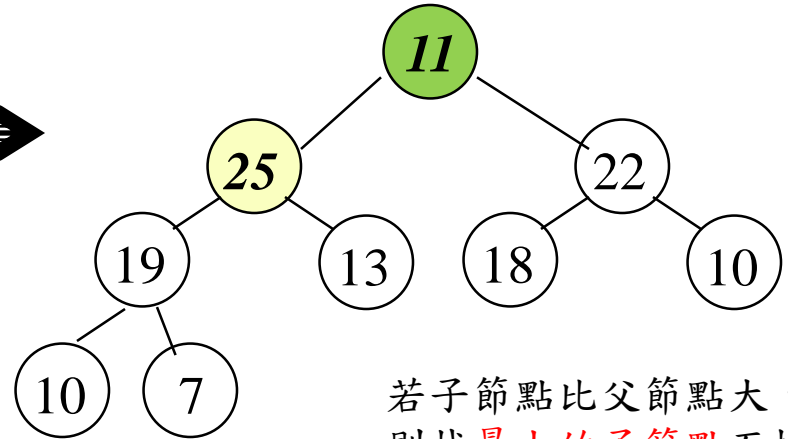


最大堆積樹：取出節點(刪除根節點)

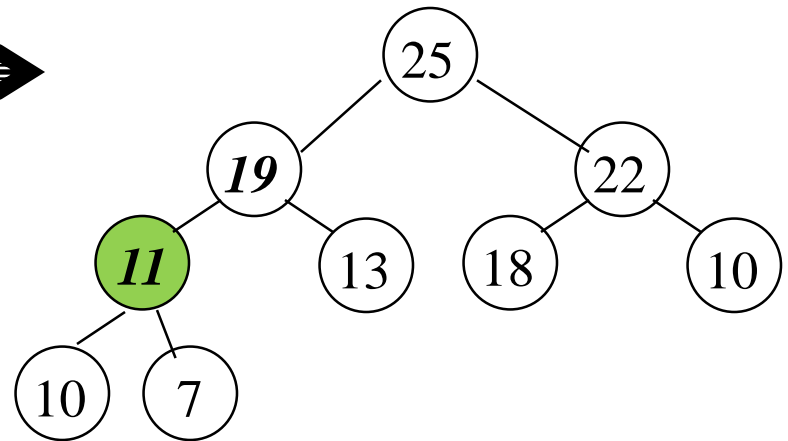
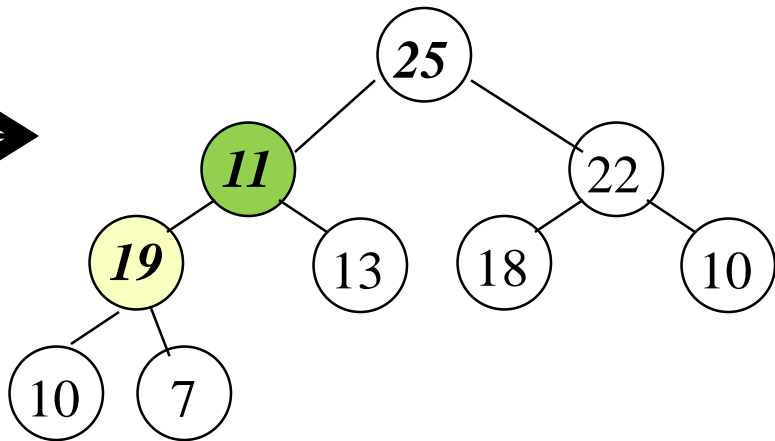
取出根節點30



以完整二元樹的最後一個節點，來取代被刪除的根節點

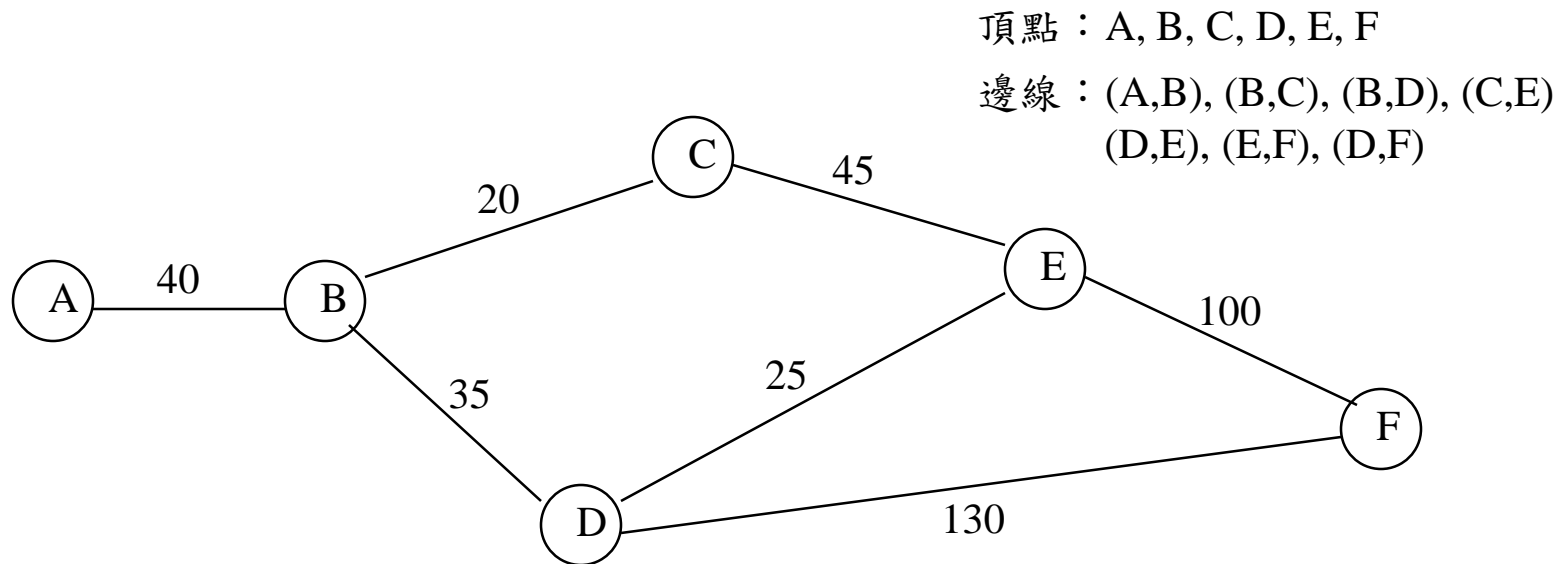


若子節點比父節點大，則找最大的子節點互換



圖形結構（Graph）

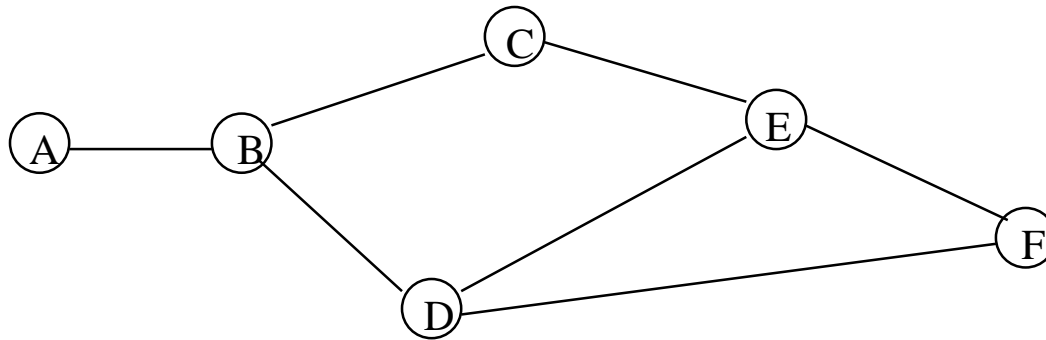
- 圖形（Graph）：一個圖形結構由兩種元素所構成
 - 頂點（vertex）
 - 邊線（edge）
- 圖形結構允許形成迴路，但樹狀結構不可形成迴路
- 例如：最短路徑問題



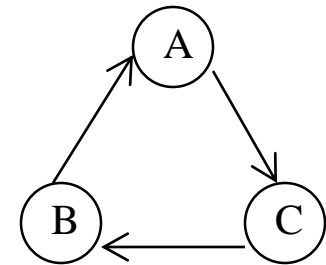
圖形的資料結構表示法

- 相鄰矩陣表示法

- 二維陣列



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 0 | 0 | 0 | 0 |
| B | 1 | 0 | 1 | 1 | 0 | 0 |
| C | 0 | 1 | 0 | 0 | 1 | 0 |
| D | 0 | 1 | 0 | 0 | 1 | 1 |
| E | 0 | 0 | 1 | 1 | 0 | 1 |
| F | 0 | 0 | 0 | 1 | 1 | 0 |

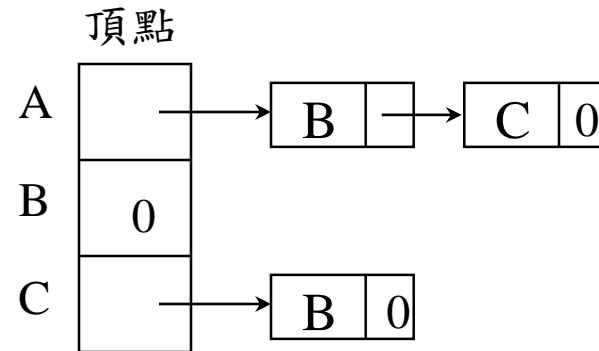
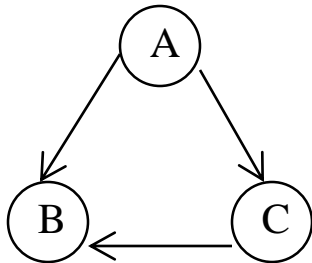
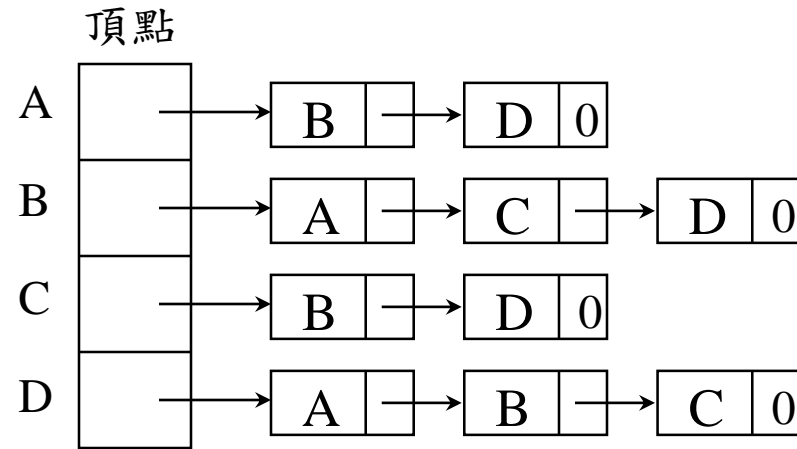
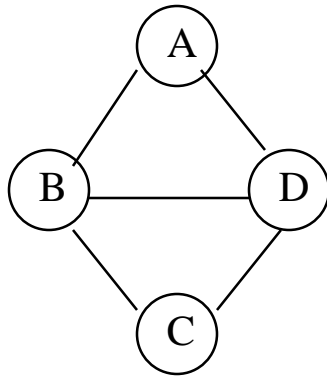


| | A | B | C |
|---|---|---|---|
| A | 0 | 0 | 1 |
| B | 1 | 0 | 0 |
| C | 0 | 1 | 0 |

圖形的資料結構表示法

● 相鄰串列表示法

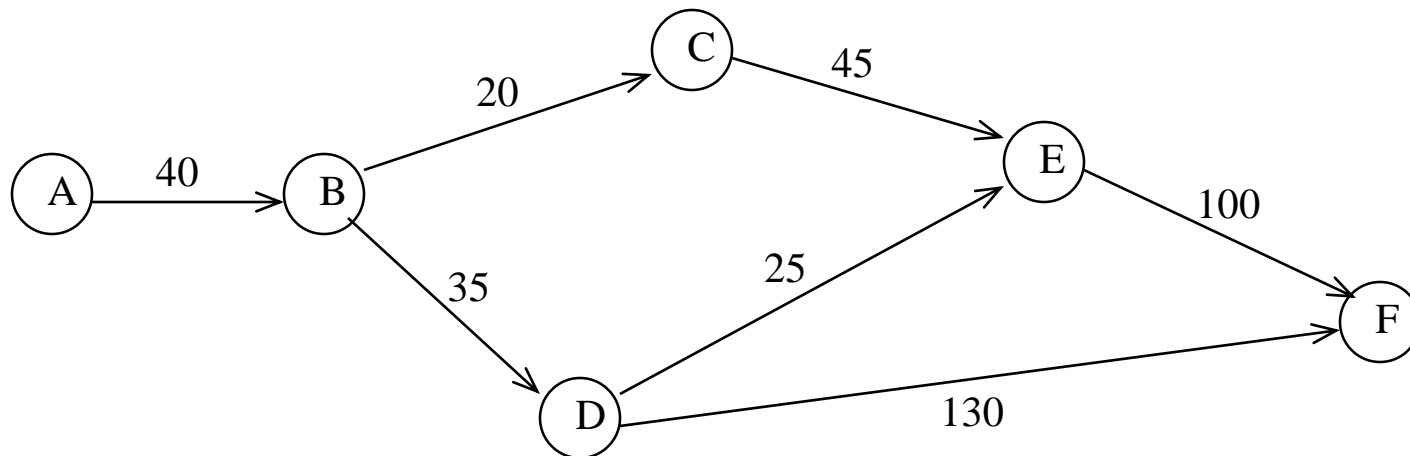
— 鏈結串列



最短路徑 (Shortest Path)

- 最短路徑

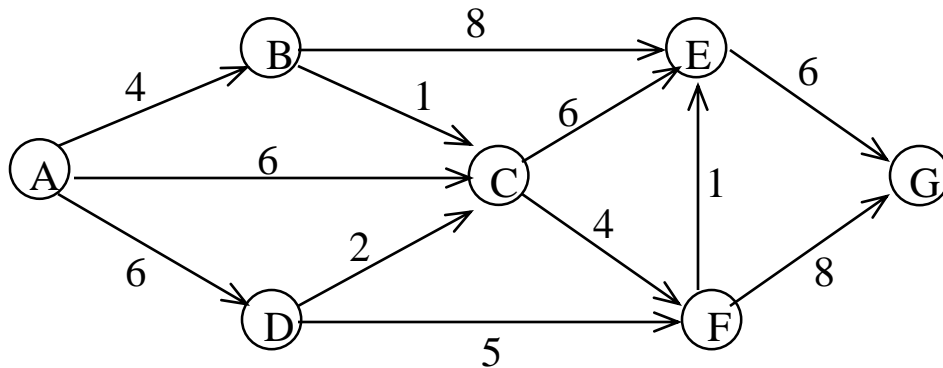
- 使用邊線，連接圖形中的兩個頂點，所需最少的成本（權重）
- 例如：下圖中A, B, D, E, F 路徑是從A 到F 的最短路徑



Dijkstra 演算法求任兩點間之最短路徑

- 1) 建立每個頂點之間的最小距離矩陣 ($D[i,j]$ 代表 i 和 j 之間的最小距離)
- 2) 針對所有的頂點 i 和 j ，每次加入一個頂點 k 來修正最小距離矩陣

$$\diamond D[i,j] = \min\{ D[i,j], D[i,k]+D[k,j] \}$$
- 3) 當所有的頂點都加入檢查過，則最小距離矩陣為任兩點間之最短路徑



| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|----------|
| A | ∞ | 4 | 6 | 6 | ∞ | ∞ | ∞ |
| B | ∞ | ∞ | 1 | ∞ | 8 | ∞ | ∞ |
| C | ∞ | ∞ | ∞ | ∞ | 6 | 4 | ∞ |
| D | ∞ | ∞ | 2 | ∞ | ∞ | 5 | ∞ |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 8 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

加入A作修正

不變

| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|----------|
| A | ∞ | 4 | 6 | 6 | ∞ | ∞ | ∞ |
| B | ∞ | ∞ | 1 | ∞ | 8 | ∞ | ∞ |
| C | ∞ | ∞ | ∞ | ∞ | 6 | 4 | ∞ |
| D | ∞ | ∞ | 2 | ∞ | ∞ | 5 | ∞ |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 8 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

加入B作修正

| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|----------|
| A | ∞ | 4 | 5 | 6 | 12 | ∞ | ∞ |
| B | ∞ | ∞ | 1 | ∞ | 8 | ∞ | ∞ |
| C | ∞ | ∞ | ∞ | ∞ | 6 | 4 | ∞ |
| D | ∞ | ∞ | 2 | ∞ | ∞ | 5 | ∞ |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 8 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

加入C作修正

| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|-----------|----------|----------|
| A | ∞ | 4 | 5 | 6 | 11 | 9 | ∞ |
| B | ∞ | ∞ | 1 | ∞ | 7 | 5 | ∞ |
| C | ∞ | ∞ | ∞ | ∞ | 6 | 4 | ∞ |
| D | ∞ | ∞ | 2 | ∞ | 8 | 5 | ∞ |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 8 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

加入D作修正

| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|----------|
| A | ∞ | 4 | 5 | 6 | 11 | 9 | ∞ |
| B | ∞ | ∞ | 1 | ∞ | 7 | 5 | ∞ |
| C | ∞ | ∞ | ∞ | ∞ | 6 | 4 | ∞ |
| D | ∞ | ∞ | 2 | ∞ | 8 | 5 | ∞ |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 8 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

加入E作修正

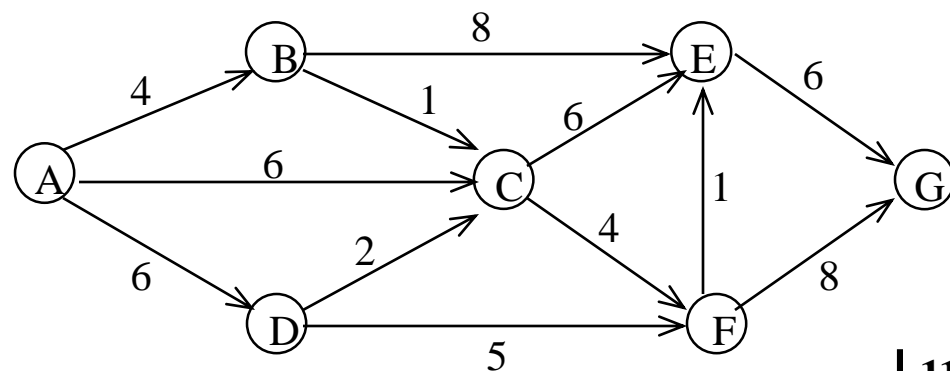
| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|-----------|
| A | ∞ | 4 | 5 | 6 | 11 | 9 | 17 |
| B | ∞ | ∞ | 1 | ∞ | 7 | 5 | 13 |
| C | ∞ | ∞ | ∞ | ∞ | 6 | 4 | 12 |
| D | ∞ | ∞ | 2 | ∞ | 8 | 5 | 14 |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 7 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

加入F作修正

| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|-----------|----------|-----------|
| A | ∞ | 4 | 5 | 6 | 10 | 9 | 16 |
| B | ∞ | ∞ | 1 | ∞ | 6 | 5 | 12 |
| C | ∞ | ∞ | ∞ | ∞ | 5 | 4 | 11 |
| D | ∞ | ∞ | 2 | ∞ | 6 | 5 | 12 |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 7 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

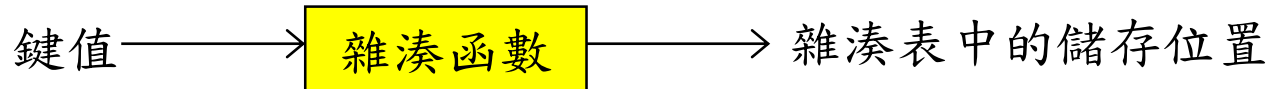
加入G

| | A | B | C | D | E | F | G |
|---|----------|----------|----------|----------|----------|----------|----------|
| A | ∞ | 4 | 5 | 6 | 10 | 9 | 16 |
| B | ∞ | ∞ | 1 | ∞ | 6 | 5 | 12 |
| C | ∞ | ∞ | ∞ | ∞ | 5 | 4 | 11 |
| D | ∞ | ∞ | 2 | ∞ | 6 | 5 | 12 |
| E | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | 6 |
| F | ∞ | ∞ | ∞ | ∞ | 1 | ∞ | 7 |
| G | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |



雜湊搜尋(Hashing)

- 存取資料時並非循序搜尋，而是代入數學函數直接求得資料的所在位置
- 雜湊函數(hash function)
 - 將鍵值(key)轉換成此資料在雜湊表(hash table)中儲存位置的數學函數



- 雜湊搜尋 VS 一般搜尋(循序搜尋或樹狀搜尋)
 - 優點
 - 所有鍵值(key)不須事先排序
 - 在沒有碰撞(collision)或溢位(overflow)的情況下，只需一個步驟就可以取出資料(將key代入雜湊函數即可得到資料的儲存位置)，搜尋速度快
 - 保密性高，若不知雜湊函數則無法擷取到資料
 - 缺點
 - 若使用靜態雜湊表(固定大小)，資料量較少時會浪費儲存空間，資料量較大時，碰撞(collision)頻率高，須處理溢位(overflow)問題，會降低搜尋效率
 - 程式設計較複雜，需要進行雜湊函數的資料轉換

雜湊搜尋的例子

- 使用雜湊表來儲存和搜尋**函數庫**中的函數，包括acos, define, float, exp, char, atan, ceil, floor, 等函數
 - 雜湊函數 $f(x) = x$ **第一個字元的對應整數值**
 - (將鍵值x的第一個字元 a~z 轉為 0~25)
 - 此雜湊表中有 26 個**桶**(bucket 0~25)，每個桶中有2個**槽** (slot 0~1)

| | Slot 0 | Slot 1 |
|-------|--------|--------|
| 0 | acos | atan |
| 1 | | |
| 2 | char | ceil |
| 3 | define | |
| 4 | exp | |
| 5 | float | floor |
| | | |
| 25 | | |

- 碰撞(collision)
 - 當鍵值 x1 代入雜湊函數後，對應值的儲存位置(**桶bucket**)已有存放**其他鍵值** x2，發生碰撞(可以儲存到下一個槽slot)
- 溢位(overflow)
 - 當某一個桶(bucket)的**所有槽 (slot)** **都已經存滿資料**，還要存放一筆新的鍵值到此桶時，發生溢位

常用的雜湊函數：中間平方法和除法

- 三種常用的雜湊函數
 - 中間平方法 (Mid-square)
 - 除法 (Division)
 - 摺疊法 (Folding)
- 中間平方法 (Mid-square)
 - 將鍵值平方後，取出中間的固定位元作為對應值
 - 例如，將鍵值平方後取出中間固定的3個位元
 - 鍵值123平方 $123^2 = 15129$ ，轉換對應值為512 (從5/2-1=1開始3個位元)
 - 鍵值9平方 $9^2 = 81$ ，轉換對應值為81 (只有2個位元，前面填入一個0)
 - 鍵值987平方 $987^2 = 974169$ ，轉換對應值為416 (從6/2-1=2開始3個位元)
- 除法 (Division)
 - 將除以一个固定質數後，所得餘數作為對應值
 - 例如，質數固定為23，則鍵值286 轉換對應值為 $286 \% 23 = 10$

常用的雜湊函數：摺疊法 (Folding)

- 摺疊法 (Folding)

- 將鍵值分為多個固定長度的分段，將每一個分段的值相加作為對應值（最後一個分段的長度會可能不足）

- 摺疊法的相加方式分為兩種

- 位移摺疊法 (shift folding)：每一個分段的值直接相加

- 鍵值 $x=123\textcolor{red}{203}241\textcolor{red}{11}220$ 分為5段 $x_1=123, x_2=203, x_3=241, x_4=112, x_5=20$

- 對應值為 $x_1+x_2+x_3+x_4+x_5 = \textcolor{red}{699}$

- 邊界摺疊法 (folding at the boundaries)：將偶數分段的值反轉後再相加

- 鍵值 $x=123\textcolor{red}{203}241\textcolor{red}{11}220$ 分為5段 $x_1=123, x_2=\textcolor{blue}{302}, x_3=241, x_4=\textcolor{blue}{211}, x_5=20$

- 對應值為 $x_1+x_2+x_3+x_4+x_5 = \textcolor{red}{897}$

- 鍵值 $x=123\textcolor{red}{456}789\textcolor{red}{01}$ 分為5段 $x_1=123, x_2=\textcolor{blue}{654}, x_3=789, x_4=\textcolor{blue}{10}$

- 對應值為 $x_1+x_2+x_3+x_4 = \textcolor{red}{1576}$

- 鍵值 $x=123\textcolor{red}{4}$ 分為2段 $x_1=123, x_2=\textcolor{blue}{4}$

- 對應值為 $x_1+x_2 = \textcolor{red}{127}$

溢位處理：線性探測法(Linear Probing)

- 線性探測法(或稱為線性開放位址)
 - 將靜態雜湊表視為**環狀空間**(桶25之後接桶0)
 - 當**溢位**發生時(要加入的桶內所有槽都存滿)，以線性方式**往下找下一個空的槽**來存放
 - 例如右表要存入 **che, zcot, drop, cart**
 - **che** => bucket 4, slot 1
 - **zcot** => bucket 1, slot 0
 - **drop** => bucket 6, slot 0
 - **cart** => bucket 6, slot 1
 - **搜尋**鍵值x的資料時
 - 代入雜湊函數 $f(x)$ ，找 $f(x)$ 值的桶
 - 若 $f(x)$ 值的桶找不到，**依序往下找** $f(x)+1$ 桶, $f(x)+2$ 桶, ... 直到找到 x 鍵值的資料或是找到空的槽 (代表 x 不存在) 為止

| | Slot 0 | Slot 1 |
|-------|---------------|--------------|
| 0 | acos | atan |
| 1 | | |
| 2 | char | ceil |
| 3 | define | deh |
| 4 | exp | |
| 5 | float | floor |
| 6 | | |
| 7 | htxt | |
| 8 | | |
| 9 | jih | |
| | | |
| 25 | zom | zeki |

溢位處理: 鏈結串列法(Chaining)

- 每個桶使用一個鏈結串列來儲存所有的資料(動態槽)
 - 除非記憶體空間滿了，否則不會發生溢位
 - 例如下面的雜湊表要存入 **che, zcot, drop, cart**

