

通識計算機程式設計期中考

臺灣大學 鄭士康 4/18/2014
試題共 7 題，兩面印製 8 頁，滿分 100



本講義除另有註明外，採創用CC姓名標示-
非商業性-相同方式分享3.0臺灣版授權釋出

1. 撰寫一或數個C#敘述達成下列要求: (假設`using System;`敘述已經包含於程式中)
 - (a) 宣告`int`變數`n`，`bool`變數`b`，`double`變數`z` (3%)
 - (b) 在螢幕顯示一行字，要求使用者輸入一個浮點數 (3%)
 - (c) 自鍵盤讀入一個浮點數，並將其值存入已宣告之`double`變數`z` (3%)
 - (d) 將已宣告設值之`double`變數`z`強制轉型為整數，存入已宣告之`int`變數`n` (3%)
 - (e) 將邏輯關係式`n >= 0 && n <= 100`設定給已宣告之`bool`變數`b` (3%)
2. 撰寫一或數個C#敘述達成下列要求: (假設`using System;`敘述已經包含於程式中)
 - (a) 將已宣告設值之`int`變數`n`用遞增算子`++`加1後，再設定(assign)給他處已宣告設值之`int`變數`m` (3%)
 - (b) 令他處已宣告設值之`int`變數`t`除以7的餘數設定給他處已宣告之`int`變數`r` (3%)
 - (c) 宣告`double`變數`d`，並設定其值為`double`變數`x`, `y`, `z`各自平方後加起來再開平方根的結果，假設`x`, `y`, `z`已於他處宣告設值 (3%)
 - (d) 宣告`int`變數`v`，利用三元運算子使其在他處已宣告設值之`double`變數`u`大於-10.0時等於1，反之則等於0 (3%)
 - (e) 宣告變數`c`為`char`型別，並令其值為「' 」字元(單引號，apostrophe) (3%)
3. 撰寫一或數個C#敘述達成下列要求: (假設`using System;`敘述已經包含於程式中)
 - (a) 先以一個敘述宣告一個亂數產生器物件`rand`，以360為其種子數；再於另一個敘述利用`rand`產生一個亂數，設定給在此宣告為`int`的變數`k` (3%)
 - (b) 宣告一個`double`常數`RESTING_VOLTAGE`，設其值為-70.0 (3%)
 - (c) 寫一個`do/while`迴圈，迴圈中自鍵盤讀入一個整數`score`，`score`大於100或小於0時繼續，否則離開迴圈。注意在迴圈前要宣告`score`為`int`變數 (3%)
 - (d) 利用`Console.ReadLine().ToCharArray()` 在一個敘述中將讀入之字串轉為字元陣列`s`，在另一個敘述中以`Array.IndexOf(s, 'g')`在字元

陣列**s**中尋找出字元**g**第一次出現的索引位置，將其值設為變數**idx**，同一敘述中也要宣告**idx**之型別為**int** (3%)

(e) 寫一個**void**函式**f**，設其唯一的形式參數為傳址之**int**參數**x**，在函式內將**x**加1 (3%)

4. 找出以下程式片段之錯誤，並予更正。

(a) (3%)

```
string info = ""; // 初值設為空字串，  
                之後會改設定為另外讀入的文字資訊
```

(b) (3%) 在 **n** 為10時不執行將其值加1之下一敘述，而直接回到**while**之處繼續執行迴圈。

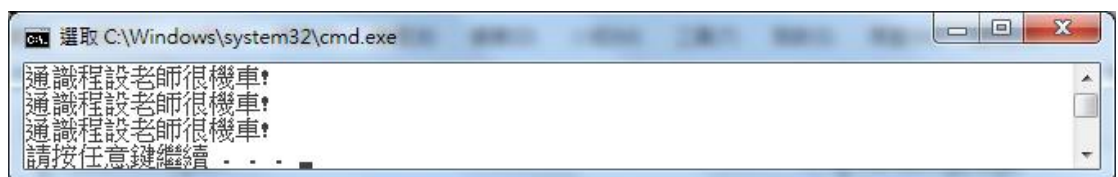
```
int n = 0;  
while(n < 100)  
{  
    if(n = 10) continue;  
    n++;  
}
```

(c) (3%) 計算 **x** 與 **y** 之間的準確比例

```
int x = 3;  
int y = 7;  
double ratio = x/y;
```

(d) (3%) 以下程式片段應印出 "通識程設老師很機車!"字樣3次，如後附之螢幕圖形

```
for(int n = 1; n < 3; n++)  
{  
    Console.WriteLine("通識程設老師很機車!");  
}
```



(e) (3%) 以下程式片段應使 **a** 變成內容為{7, 8, 9}之新整數陣列

```
class Program
{
    static void Main(string[] args)
    {
        int[] a = { 1, 2, 3, 4, 5 };
        ModifyArray(a);
    }

    static void ModifyArray(int[] a)
    {
        a = new int[] { 7, 8, 9 };
    }
}
```

5. 試寫出下列程式的螢幕輸出 (5 %)

```
using System;

namespace Problem5
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] a = { 2, 6, 4, 8 };
            int i;
            Console.Write("original a = [ ");
            for (i = 0; i < a.Length-1; i++)
            {
                Console.Write(" {0}, ", a[i]);
            }
            Console.WriteLine(" {0} ]", a[a.Length-1]);
            BS(a);
            Console.Write("modified a = [ ");
```

```

        for (i = 0; i < a.Length-1; i++)
        {
            Console.Write(" {0}, ", a[i]);
        }
        Console.WriteLine(" {0} ]", a[a.Length-1]);
    }

    static void BS(int[] b)
    {
        for (int pass = 1; pass < b.Length; pass++)
        {
            for(int i = 0; i < b.Length - 1; i++)
            {
                if (b[i] > b[i + 1])
                    SAPOE(b, i);
            }
        }
    }

    static void SAPOE(int[] c, int idx)
    {
        int hold = c[idx];
        c[idx] = c[idx + 1];
        c[idx + 1] = hold;
    }
}

```

6. 試寫出下列程式的螢幕輸出 (10 %)

```

using System;
namespace Problem6
{
    class Program
    {
        static void Main(string[] args)
        {

```

```

int[,] trainingPattern = { { 1, 1 }, { 1, 0 } };
int[] input = new int[2];
double[,] w = { { 0.5, 0.2 }, { -0.3, 0.7 } };
int[] output = new int[2];
int i;
double rate = 0.1;
for (i = 0; i < 2; i++)
{
    input[0] = trainingPattern[i, 0];
    input[1] = trainingPattern[i, 1];
    output = PerceptronOutput(w, input);
    LearnPerceptronWeight(rate, w,
        input, output);
    Console.WriteLine(
        "Training Pattern {0}: [ {1}, {2} ]",
        i + 1, input[0], input[1]);
    Console.WriteLine("Output = [ {0}, {1} ] ",
        output[0], output[1]);
    Console.WriteLine();
}
int[] test = new int[2] { 0, 1 };
output = PerceptronOutput(w, test);
Console.WriteLine(
    "Test Pattern : [ {0}, {1} ]",
    test[0], test[1]);
Console.WriteLine("Output = [ {0}, {1} ]",
    output[0], output[1]);
Console.WriteLine();
}

static int[] PerceptronOutput(double[,] w,
    int[] input )
{
    double sum;
    int i;
    int j;
    int[] result = new int[w.GetUpperBound(0)+1];
    for (i = 0; i < w.GetUpperBound(0) + 1; i++)

```

```

    {
        sum = 0.0;
        for (j = 0; j < w.GetUpperBound(1) + 1; j++)
        {
            sum += w[i, j] * input[j];
        }
        result[i] = (sum > 0.0) ? 1 : 0;
    }
    return result;
}

static void LearnPerceptronWeight(double rate,
    double[,] w, int[] input, int[] output)
{
    int i;
    int j;
    for(i=0; i < w.GetUpperBound(0) + 1; i++)
    {
        for (j = 0; j < w.GetUpperBound(1) + 1; j++)
        {
            w[i, j] += rate * output[i] * input[j];
        }
    }
}
}
}

```

7. 賽局理論(game theory)中,「囚徒困境」(prisoners' dilemma)的經典描述如下:

警方逮捕甲、乙兩名嫌疑犯,但沒有足夠證據指控二人有罪。於是警方分開囚禁嫌疑犯,分別和二人見面,並向雙方提供以下相同的選擇:

- 若一人認罪並作證檢控對方(「背叛」, betray),而對方保持沉默,此人將即時獲釋,沉默者將判監 10 年。
- 若二人都保持沉默(「合作」, cooperate),則二人同樣判監半年。
- 若二人都互相檢舉(互相「背叛」),則二人同樣判監 2 年。

用表格概述為

	甲沉默（合作）	甲認罪（背叛）
乙沉默（合作）	二人同服刑半年	甲即時獲釋；乙服刑 10 年
乙認罪（背叛）	甲服刑 10 年；乙即時獲釋	二人同服刑 2 年

困境中兩名理性囚徒的選擇：

- 若對方沉默、我背叛會讓我獲釋，所以會選擇背叛。
- 若對方背叛指控我，我也要指控對方才能得到較低的刑期，所以也是會選擇背叛。

二人面對的情況一樣，所以二人的理性思考都會得出相同的結論——選擇背叛。參與者都背叛對方，結果二人同樣服刑 2 年。這顯然不是顧及團體利益的最優解決方案。以全體利益而言，如果兩個參與者都合作保持沉默，兩人都只會被判刑半年，總體利益更高，結果也比兩人背叛對方、判刑 2 年的情況較佳。

現實中無論是人類社會或大自然都可以找到類似「囚徒困境」的例子，但是這些例子中，雙方可以有許多次的互動，並且記住他們以前的對抗結果，稱作「重複囚徒困境」（iterated prisoners' dilemma）。社會科學中的實驗經濟學、政治學和社會學，以及自然科學的動物行為學、進化生物學、決策心理學等學科，有許多問題都可以用「重複囚徒困境」分析，模擬生物面對無止境的「囚徒困境」博弈(game)。

Robert Axelrod 在其著作 *The Evolution of Cooperation* (1984)敘述進行其研究時，邀請全世界的學術同行來設計各種計算機策略，在「重複囚徒困境」競賽中互相競爭。參賽的策略在演算法的複雜性、最初的對抗、寬恕的能力等等有很廣泛的差異。Robert Axelrod 發現：當每個選擇不同策略的參與者對抗重複了很長時間之後，從利己的角度來判斷，最終「貪婪」策略趨向於減少，而比較「利他」的策略反而較常被採用。他用這個結論說明：通過自然選擇，一種利他行為的機制可能從最初純粹的自私機制演化而來。

在這樣的比賽中，最佳確定性策略通常認為是 Anatol Rapoport 的「以牙還牙」(tit for tat, 亦即"equivalent retaliation")，是所有參賽策略中最簡單的，只包含了四行 BASIC 語言，並且贏得了比賽。這個策略只不過是在重複博弈的開頭合作，然後，採取你的對手前一回合的策略。更好些的策略是「寬恕地以牙還牙」(tit for tat with forgiveness)。當你的對手背叛，在下一回合中你無論如何要以小機率（大約是 1%-5%）時而合作一下，這是考慮到偶爾要從循環背叛的受騙中復原。

(以上資料取材自維基百科網頁:

<http://zh.wikipedia.org/wiki/%E5%9B%9A%E5%BE%92%E5%9B%B0%E5%A2%83>

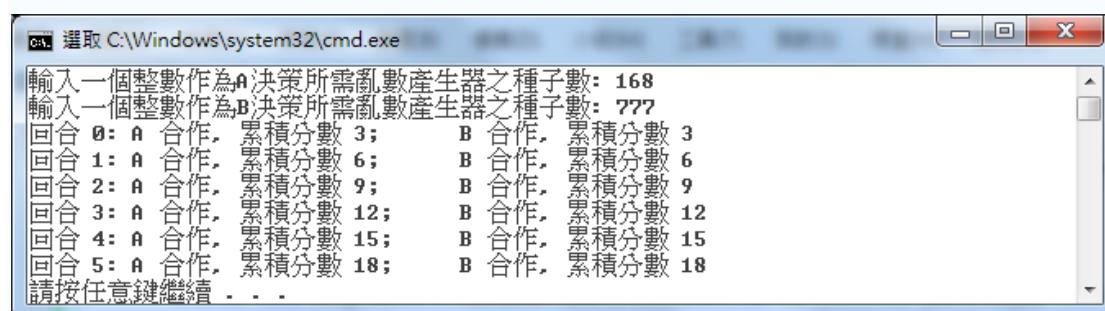
網頁中有更完整的說明及許多領域的應用範例)

本題請你撰寫一個 C# 程式，模擬「重複囚徒困境」的狀況，起始雙方均合作(算是第 0 回合)，接著對抗 N 個回合。由第 0 回合開始，在螢幕顯示每一回合雙方採用的策略及由開始到這一回合的累計得分。假設 N 是一個 1(含)與 1000(含)之間的隨機亂數(因此雙方事先不知道要進行多少回合，亂數產生器可以不設種子數(seed)，讓計算機自行決定，使程式每次執行結果都不相同)，且對抗雙方(代號為 A, B)都採用「寬恕地以牙還牙」策略，寬恕對手背叛的機率 A 為 5%，B 為 2%，而其每一回合的酬報矩陣(payoff matrix, 顯示結果的表格)如下:

	B 合作	B 背叛
A 合作	A, B 均得 3 分	A 得 0 分, B 得 5 分
A 背叛	A 得 5 分, B 得 0 分	A, B 均得 1 分

提示: 對手背叛時，可以產生一個 0 到 99 之間的隨機亂數，如果這個亂數是 0、1，就寬恕對方(選取「合作」)，否則就選「背叛」(以牙還牙)，這樣即可以用 2% 的機率寬恕對方。類似的作法可以得到 5% 的寬恕機率。兩個對手的亂數產生器種子數可隨意用兩個你自己選定的不同整數。

程式螢幕互動範例:



```
C:\Windows\system32\cmd.exe
輸入一個整數作為A決策所需亂數產生器之種子數: 168
輸入一個整數作為B決策所需亂數產生器之種子數: 777
回合 0: A 合作, 累積分數 3;    B 合作, 累積分數 3
回合 1: A 合作, 累積分數 6;    B 合作, 累積分數 6
回合 2: A 合作, 累積分數 9;    B 合作, 累積分數 9
回合 3: A 合作, 累積分數 12;   B 合作, 累積分數 12
回合 4: A 合作, 累積分數 15;   B 合作, 累積分數 15
回合 5: A 合作, 累積分數 18;   B 合作, 累積分數 18
請按任意鍵繼續 . . .
```

本題滿分 25 分，全部程式集中寫成一個大 **Main** 函式，不區分函式者，最高得 20 分；善用函式，乃至尚未教到的物件導向程式設計(object-oriented programming)者，最高得 23 分；能利用虛擬碼或流程圖思考，適當劃分函式或類別(class)者，最高得 25 分(使用虛擬碼)或 24 分(使用流程圖)。(25%)