HW6

班級	姓名	學號	日期
四機械四乙	吳宇昕	B10831020	12/10/2022

Q0 Main2Tuple

sorce code and replit

終端機輸出

```
Student ID: B10831020
12/10/2022 4:56:26 PM
[觀-照-知-悟-證-行]::ME5015 工程領域的程式語言實務

Please enter the first number
23.11
Please enter the seconde number
57
sum = 80.11
avg = 40.055

Loop count 3

Loop= 0, 38, 490
Loop= 1, 46, 720
Loop= 2, 25, 110
```

將Get2Values寫成獨立函式，去除零散的Console.WriteLine()程式碼。在Main產生StringBuilder物件，並搭配ref關鍵字傳進此函式。

```
52      //  GOAL:: GenerateRandomNumbers --
53      //  TODO: make it complete as a new Method:
          1 reference
54      public static void Get2Values(int ceiling, int cCount, ref System.Text.StringBuilder _sb){
55        Random rnd = new Random();
56        int inum1, inum2;
57        _sb.AppendLine($"\nLoop count {cCount}\n") ; //TODO
58        for (var idx =0; idx < cCount ;idx++) {
59          inum1 = rnd.Next() % ceiling;
60          inum2 = rnd.Next() % ceiling *10;
61          _sb.AppendLine(String.Format("Loop= {2:G3}, {0:G4}, {1:G4}", inum1, inum2, idx ));
62        }
63      }
```

心得

C#的Tuple用法很多元。可以幫每個tuple內的元素命名，形成類似struct或dictionary以欄位名稱索引的資料結構。

然而其方便性仍遠不如Python的tuple。不清楚為什麼，C#的tuple不支援foreach loop，也不支援以index取值，必須要用`.item1`, `.item2`的方式取出第一與第二個元素值。

像是這樣的語法，在C#行不通

```
var t = (23, 66, 22); // creating a tuple<int, int, int>
Console.Write(t[0]);  // unable to get the value 23 by indexing 0
```

必須要寫成這樣

```
var t = (23, 66, 22); // creating a tuple<int, int, int>
Console.Write(t.Item1);  // get the value 23 by ".Item1"
```

tuple不支援indexing跟foreach迴圈，恐怕不適合被迴圈走訪。或許C#的Tuple設計上並不打算讓我們這樣做？或是其實有正確的作法，但是我還沒學到。目前看起來，它最適合打包函式的回傳值。這讓C#的函式看起來可以像Python的函式一次回傳多個值，在接收回傳值的地方利用類似Python unpacking語法，輕鬆拆封tuple。

Q3A

sorce code and replit

終端機輸出

```
Student ID: B10831020
The two points fartest apart are (-9.5, -2.1) and (10.3, -2.1)
With distance 19.800
```

計算最長距離

題目給7個點的x,y座標，求最遠兩點的距離。求解過程如下：

1. 找出擁有最大與最小x,y座標值的四個點，最遠距離一定是此四點其中兩點距離
2. 設一變數紀錄最長距離
3. 計算此四點兩兩之間的距離，若當下的兩點距離大於紀錄的最長距離，就取而代之

心得

計算歐式距離需要開根號，耗費較多計算資源，應盡可能降低開根號次數。若要計算每一個點與其他點之間的距離，須至少開C(7,2)次根號。但是確定最大距離一定發生在四個邊界點之間，只需要計算四個邊界點兩兩之間的距離，開C(4,2)次根號就夠了。若題目加入更多點的座標，不會增加開根號次數。

或許這個題目還有更好的解法，進一步減少計算成本，但目前我只想到這個做法。

Q3B

sorce code and replit

終端機輸出

Student	Grades			Stu Avg
0	90	90	80	86.667
1	80	80	70	76.667
2	50	60	70	60.000
3	40	80	80	66.667
4	60	60	70	63.333
5	70	80	70	73.333
6	90	60	50	66.667
7	30	80	60	56.667
8	60	60	50	56.667
Avg	63.333	72.222	66.667	

心得

C#有個很好用的關鍵字`readonly`，讓一個class attribute的值經初始化後便改為唯讀，不可變更。這比C++的`const`關鍵字好用，因為一個const member沒辦法初始化賦值。 C#好像不讓我們把的class member設為const，若要一個class member值固定不變，必須用readonly。因此這題我把學生的成績設為`readonly int[,]`，放在`class Program`裡面。

```
59          private static readonly int[,] sGrades =
60          {
61              {90, 90, 80},
62              {80, 80, 70},
63              {50, 60, 70},
64              {40, 80, 80},
65              {60, 60, 70},
66              {70, 80, 70},
67              {90, 60, 50},
68              {30, 80, 60},
69              {60, 60, 50}
70          };
```

Q4

sorce code main.cs Deck.cs Card.cs

and

replit

三份cs檔分別包含`class Program`、`class Deck`及`class card`，皆屬於`namespace Q4`

終端機輸出

```
Studnet ID: B10831020

Before shuffling
| A--berry     A--flower    A--diamond   A--heart    |
| 2--berry     2--flower    2--diamond   2--heart    |
| 3--berry     3--flower    3--diamond   3--heart    |
| 4--berry     4--flower    4--diamond   4--heart    |
| 5--berry     5--flower    5--diamond   5--heart    |
| 6--berry     6--flower    6--diamond   6--heart    |
| 7--berry     7--flower    7--diamond   7--heart    |
| 8--berry     8--flower    8--diamond   8--heart    |
| 9--berry     9--flower    9--diamond   9--heart    |
|10--berry    10--flower   10--diamond  10--heart    |
| J--berry     J--flower    J--diamond   J--heart    |
| Q--berry     Q--flower    Q--diamond   Q--heart    |
| K--berry     K--flower    K--diamond   K--heart    |

After shuffling
| 4--berry     4--heart     J--diamond   A--heart    |
| A--diamond   6--flower    5--diamond   2--heart    |
| 7--berry     7--diamond   8--berry    10--diamond  |
| 3--berry     Q--diamond   2--flower    8--flower   |
|10--heart     Q--heart     8--heart     4--flower   |
| Q--flower    Q--berry     5--berry     4--diamond  |
| 9--flower    6--diamond   7--heart     K--berry    |
| 5--flower    J--flower   10--berry     A--berry    |
| K--heart    10--flower    9--diamond   9--heart    |
| 3--diamond   2--berry     J--berry     6--heart    |
| 8--diamond   A--flower    6--berry     J--heart    |
| 7--flower    2--diamond   3--heart     3--flower   |
| 5--heart     K--flower    K--diamond   9--berry    |
```

自定義Card物件

```
namespace Q4
{
    class Card
    {
        public readonly static string[] sSuit = {"berry", "flower", "diamond",
"heart"};
        public readonly static string[] sNumber = {"A", "2", "3", "4", "5", "6",
"7", "8", "9", "10", "J", "Q", "K"};
        public readonly int SuitIdx;
        public readonly int numberIdx;

        public Card(int _suitIdx, int _numberIdx)
        {
            this.SuitIdx = _suitIdx;
            this.numberIdx = _numberIdx;
        }
```

```
    }
}
```

每張牌都有一個花色與一個數值，兩者都是string。然而，過去似乎聽說string是指向heap的char pointer，在程式裡生成過多string物件容易使記憶體零散。因此，每張牌的花色與數值欄位我並沒有用string的方式儲存，以int儲存，作為索引另外兩個static string array的索引值。

不知道這樣做是否真的可以提升程式效能，減少記憶體零散，或是只是我自找麻煩？後續要印出一張牌的內容時，增加了不少困擾，要寫成這樣：

```
Card c = new Card(3, 11);
Console.WriteLine($"{sSuit[c.SuitIdx]}--{sNumber[c.numberIdx]}")
// diamond--J
```

而不是這樣：

```
Card c = new Card(3, 11);
Console.WriteLine($"{c.Suit}--{c.Number}");
// diamond--J
```

我的寫法大幅減少可讀性，不知道是否真的能像我想像的一樣增進效能。