

夏季學院通識計算機程式設計期末考參考解答

8/14/2020

1.

(a) 宣告整數常數 **N = 10**，再宣告一個長度為 **N** 的一維整數陣列 **fib** (3%)

Answer:

```
const int N = 10;
int[] fib = new int[N];
```

(b) 寫一個迴圈，將 **fib** 的每個元素都設為 **0** (3%)

Answer:

```
for(int n = 0; n < N; ++n)
{
    fib[n] = 0;
}
```

(c) 設定 **fib[1]** 為 **1**，再寫一個迴圈，迴圈變數 **n** 從 **2** 開始，在迴圈內設定 **fib[n]** 為其前兩項 **fib[n-1]**、**fib[n-2]** 之和 (3%)

Answer:

```
fib[1] = 1;
for(int n = 2; n < N; ++n)
{
    fib[n] = fib[n-2] + fib[n-1];
}
```

(d) 宣告並設定一個整數變數 **match** 之值為 **21**，利用 C# 提供的函式 **Array.IndexOf** 在陣列 **fib** 中尋找與 **match** 相同之元素的索引。如果沒找到，**Array.IndexOf** 會傳回 **-1** (3%)

Answer:

```
int match = 21;
int idx = Array.IndexOf(fib, match);
```

(e) 利用 C# 提供的函式 **Array.Reverse**，將陣列 **fib** 的元素倒過來排列 (3%)

Answer:

```
Array.Reverse(fib);
```

2.

- (a) 宣告一個二維陣列 **trans**，同時設定初值，用以表示如 表1 所示：The Peach Blossom Spring Town、Gotham City 兩地區，今年第一季(一到三月)的房地產交易值 (3%)

表1. 今年第一季(一到三月)兩地區的房地產交易值(單位:億元)

	The Peach Blossom Spring Town	Gotham City
January	135	246
February	95	183
March	120	212

Answer:

```
int[,] trans =  
{  
    {135, 246},  
    {95, 183},  
    {120, 212}  
};
```

- (b) 宣告整數變數 **maxi** 和 **maxj**，並利用 C# 提供的函數 **GetUpperBound**，分別設定 **maxi** 和 **maxj** 為陣列 **trans** 對應的列索引(row index)和行索引(column index)最大值 (3%)

Answer:

```
int maxi = trans.GetUpperBound(0);  
int maxj = trans.GetUpperBound(1);
```

- (c) 宣告整數陣列 **rowSum**，以迴圈設定其第 **i** 個元素為第 **i** 個月中，兩個地區的交易量總和 (3%)

Answer:

```
int[] rowSum = new int[maxi+1];  
for(int i = 0; i <= maxi; ++i)  
{  
    rowSum[i] = 0;  
    for(int j = 0; j <= maxj; ++j)  
    {  
        rowSum[i] += trans[i, j];  
    }  
}
```

- (d) 宣告整數陣列 **colSum**，以迴圈設定其第 **j** 個元素為第 **j** 個地區在第一季三個月的交易量總和 (3%)

Answer:

```

int[] colSum = new int[maxj+1];
for(int j = 0; j <= maxj; ++j)
{
    colSum[j] = 0;
    for(int i = 0; i <= maxi; ++i)
    {
        colSum[j] += trans[i, j];
    }
}

```

- (e) 宣告整數變數 **total**，以 **foreach** 迴圈敘述，設定其值為兩個地區第一季的所有房地產交易量總和 (3%)

Answer:

```

int total = 0;
foreach(int tr in trans)
{
    total += tr;
}

```

3.

- (a) 宣告一個亂數產生器 **rand**，其種子數 **seed** 為 **777** (3%)

Answer:

```

Random rand = new Random(777);

```

- (b) 宣告一個 **static int** 函數 **ThrowOctahedronDice**，以一個亂數產生器 **rand** 為引數，大括弧中的內容空白 (3%)

Answer:

```

static int ThrowOctahedronDice(Random rand)
{
}

```

- (c) 完成函式 **ThrowOctahedronDice** 的內容：以引數 **rand** 產生一個隨機正整數，求此正整數除以 **8** 的餘數加 **1**，就可以得到一個介於 **1** 到 **8** 的整數，作為傳回值，代表某次丟擲八面骰子時，向上正面顯現的點數 (3%)

Answer:

```

int faceValue = rand.Next() % 8 + 1;
return faceValue;

```

- (d) 在主程式 **Main** 中，宣告整數常數 **N_VALUES = 8** 代表正面點數的個數。宣告長度為 **N_VALUES** 的整數陣列 **accum**，用來累計各個點數出現的次數。再其次宣告整數常數 **N_TRIALS = 10000** (3%)

Answer:

```

const int N_VALUES = 8;
int[] accum = new int[N_VALUES];
const int N_TRIALS = 10000;

```

- (e) 寫一個迴圈，呼叫 **ThrowOctahedronDice** 函式 **N_TRIALS** 次，在陣

列 **accum** 的對應元素累計各點數出現的次數。完成迴圈後，用 **Console.WriteLine** 印出結果如圖 1 (3%)

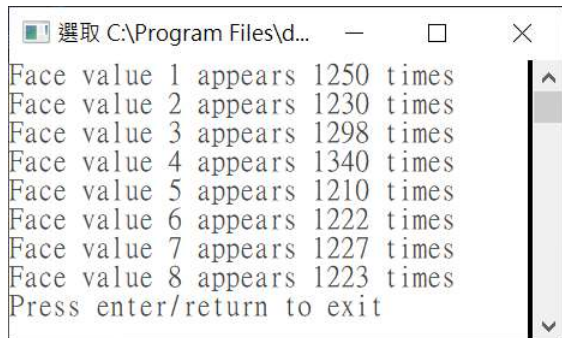


圖1. 丟擲正八面體骰子(octahedron dice) 10,000次後，點數 1 到 8 的出現次數分布

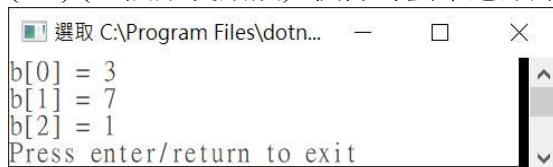
Answer:

```
int faceValue;
for(int trial = 0; trial < N_TRIALS; ++trial)
{
    faceValue = ThrowOctahedronDice(rand);
    accum[faceValue-1]++;
}

for(int v = 0; v < N_VALUES; ++v)
{
    Console.WriteLine("Face value {0} appears {1} times", v+1, accum[v]);
}
```

4.

(a) (3%) (一個語義錯誤) 執行時螢幕應顯示



```
int[] a = {3, 7, 1};
int[] b = new int[3];
b = a;
Array.Reverse(a);
for(int idx = 0; idx < 3; ++idx)
{
    Console.WriteLine("b[" + idx + "] = " + b[idx]);
}
```

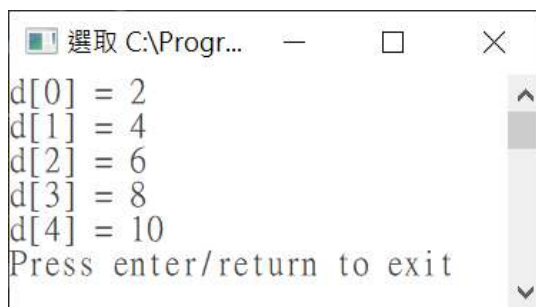
Answer:

由於 **b = a;** 代表直接將陣列 **a** 的參考(reference)設給陣列 **b**, 因此兩者成為佔據相同記憶區域的陣列。接著當陣列 **a** 被倒排(Reverse), **a** 的內容成為{1, 7, 3}, 由於是同一個記憶區域, **b** 的內容也同時成為 {1, 7, 3}, 與螢幕截圖要求不同。螢幕所顯示的 **b** 內容是 **a** 被倒排前的

結果。所以最簡單的方法是利用 `Array.Copy` 函式，取代 `b = a;` 讓 `a` 的內容可以設給 `b`，但是 `Array.Copy` 會通知作業系統，另外開闢一塊記憶區給陣列 `b`。這就不會產生倒排陣列 `a`，卻同時讓陣列 `b`，也跟著被倒排的副作用(side effect)。更正後的程式如下：

```
int[] a = {3, 7, 1};
int[] b = new int[3];
b = a; // delete
Array.Copy(a, b, 3); // insert
Array.Reverse(a);
for(int idx = 0; idx < 3; ++idx)
{
    Console.WriteLine("b[" + idx + "] = " + b[idx]);
}
```

(b) (3%) (一個語義錯誤) 執行時螢幕應顯示



```
int[] d = {1, 2, 3, 4, 5};
for(int idx = 0; idx <= d.Length; ++idx)
{
    d[idx] *= 2;
    Console.WriteLine("d[" + idx + "] = " + d[idx]);
}
```

Answer:

陣列 `d` 的有效索引為 `0, 1, ..., 4`，而 `d.Length` 代表陣列 `d` 的長度 `5`，所以迴圈會多執行一次到 `idx = 5`；使 `d[idx]` 對應的記憶體位置超過了作業系統分配給陣列 `d` 的區域，產生例外(exception，也稱為 run-time error)。原程式只要把迴圈變數上限處的 `<=` 符號，改成 `<` 即可。更正程式碼如下：

```
int[] d = {1, 2, 3, 4, 5};
for(int idx = 0; idx < d.Length; ++idx)
```

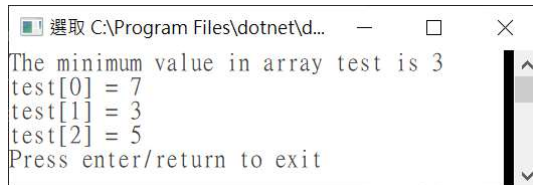
```

{
    d[idx] *= 2;

    Console.WriteLine("d[" + idx + "] = " + d[idx]);
}

```

(c) (3%) (一個語義錯誤) 執行時螢幕應顯示



```

static int Minimum(int[] x)
{
    Array.Sort(x);
    return x[0];
}

static void Main(string[] args)
{
    int[] test = {7, 3, 5};
    int min = Minimum(test);
    Console.WriteLine(
        "The minimum value in array test is " + min);
    for(int idx = 0; idx < 3; ++idx)
    {
        Console.WriteLine("test[" + idx + "] = " + test[idx]);
    }
}

```

Answer:

函式 **Minimum** 中的引數為陣列 **x**，而且為傳值呼叫。傳值呼叫會將真實引數，複製後傳送到被呼叫的函式。現在引數是一個陣列，如果各種情況都要將內容元素逐一複製，非常沒有效率。所以陣列的傳值呼叫，僅是將陣列參考直接複製給被呼叫函式，也因此函式內對陣列的修改，也會影響到原陣列。如果此種副作用必須避免，如本題要求，僅改為傳址呼叫(傳送陣列參考的地址)沒有用，而應將函式輸入陣列引數，在函式中，以**Array.Copy**複製一份，並改在複製的陣列運算，就可以避免以上副作用。更正的程式碼如下：

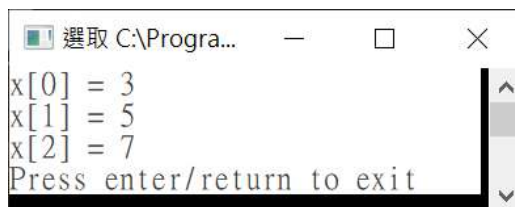
```

static int Minimum(int[] x)
{
    Array.Sort(x); // delete
    return x[0]; // delete
    int[] y = new int[x.Length]; // insert
    Array.Copy(x, y, x.Length); // insert
    Array.Sort(y); // insert
    return y[0]; // insert
}

static void Main(string[] args)
{
    int[] test = {7, 3, 5};
    int min = Minimum(test);
    Console.WriteLine(
        "The minimum value in array test is " + min);
    for(int idx = 0; idx < 3; ++idx)
    {
        Console.WriteLine("test[" + idx + "] = " + test[idx]);
    }
}

```

(d) (3%) (一個語義錯誤) 執行時螢幕應顯示如下:



```

static void Increment(int x)
{
    ++x;
}

static void Main(string[] args)
{
    int[] s = {2, 4, 6};
    for(int idx = 0; idx < 3; ++idx)
    {
        Increment(s[idx]);
        Console.WriteLine("x[" + idx + "] = " + s[idx]);
    }
}

```

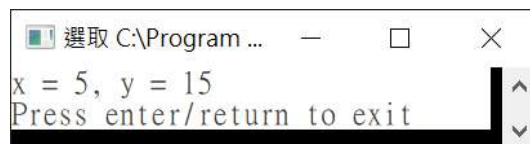
Answer:

函式 **Increment** 是傳值呼叫，因此會將輸入引數 **x** 複製一份，對複製的變數進行運算，不會改變原先的引數 **x** 之值。本題主程式以陣列元素 **s[idx]** 作為引數呼叫 **Increment**，由於並非傳送整個 **x** 陣列的參考(reference)，所以實際上是將陣列元素 **s[idx]** 的數值複製一份，來進行 **Increment** 內的計算，無法因此改變陣列元素 **s[idx]** 的內容。本題情況下，我們需要在函式 **Increment** 中改變輸入引數對應的原變數，所以將函式 **Increment** 改為傳址呼叫，就可以解決這個問題。更正的程式如下：

```
static void Increment(ref int x)
{
    ++x;
}

static void Main(string[] args)
{
    int[] s = {2, 4, 6};
    for(int idx = 0; idx < 3; ++idx)
    {
        Increment(ref s[idx]);
        Console.WriteLine("x[" + idx + "] = " + s[idx]);
    }
}
```

(e) (3%) (一個語法錯誤) 執行時螢幕應顯示如下：



```
static void TripleX(int x, int y)
{
    y = 3*x;
}

static void Main(string[] args)
{
    int x = 5;
    int y;
    Triple(x, y);
    Console.WriteLine("x = {0}, y = {1}", x, y);
}
```


Answer:

主程式中，呼叫函式 **Triple** 時，變數 **y** 並沒有給定初值，無法用來呼叫，形成語法錯誤，不能完成建置。解決的方法，可以在宣告時，就設定 **y** 的初值，如 **int y = 0;**。不過如考慮到 **y** 的數值是由 **Triple** 產生，呼叫 **Triple** 前，設定其數值並不合理。此時可以使用 **out** 關鍵字，表示 **y** 由某函式產生即可。更正的程式如下：

```
static void TripleX(int x, out int y)
{
    y = 3*x;
}

static void Main(string[] args)
{
    int x = 5;
    int y;
    Triple(x, out y);
    Console.WriteLine("x = {0}, y = {1}", x, y);
}
```

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)
        {
            const int N = 25;
            const int M = 7;
            int n = 0;
            for(int i = 1; i < N; ++i)
            {
                if(i / M * M == i) ++n;
                Console.WriteLine("i = {0}, n = {1}", i, n);
            }
        }
    }
}
```

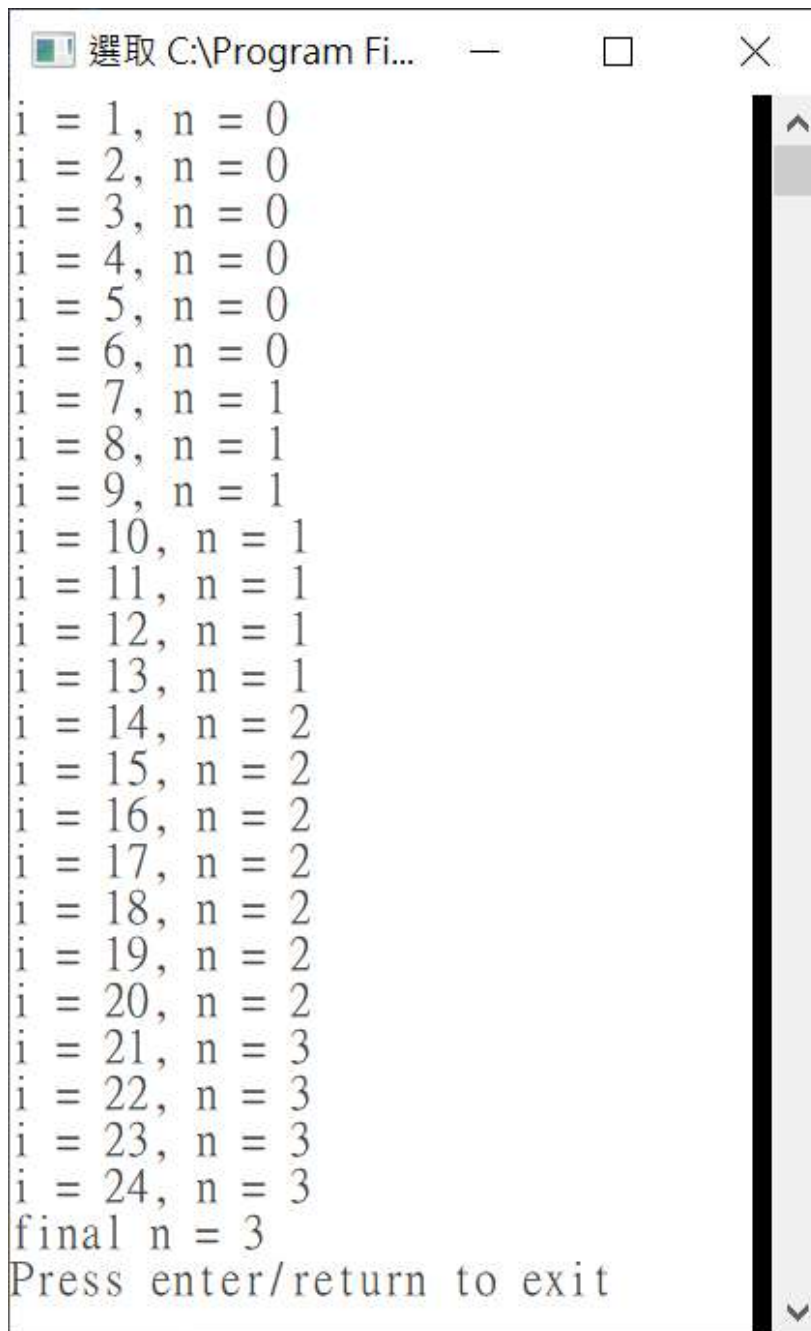
```

        Console.WriteLine("final n = " + n);

        Console.WriteLine("Press enter/return to exit");
        Console.ReadLine();
    }
}

```

Answer:



```

i = 1, n = 0
i = 2, n = 0
i = 3, n = 0
i = 4, n = 0
i = 5, n = 0
i = 6, n = 0
i = 7, n = 1
i = 8, n = 1
i = 9, n = 1
i = 10, n = 1
i = 11, n = 1
i = 12, n = 1
i = 13, n = 1
i = 14, n = 2
i = 15, n = 2
i = 16, n = 2
i = 17, n = 2
i = 18, n = 2
i = 19, n = 2
i = 20, n = 2
i = 21, n = 3
i = 22, n = 3
i = 23, n = 3
i = 24, n = 3
final n = 3
Press enter/return to exit

```

6. 試寫出下列程式的螢幕輸出 (修改自

<https://www.eximiaco.tech/en/2019/11/17/computing-the-levenshtein-edit-distance-of-two-strings-using-c/>) (10 %)

```
using System;

namespace Problem6
{
    class Program
    {
        static int Minimum(int e1, int e2, int e3)
        {
            int r1 = Math.Min(e1, e2);
            int result = Math.Min(r1, e3);
            return result;
        }

        static int ComputeTheDistanceBetween(string s1, string s2)
        {
            char[] first = s1.ToCharArray();
            char[] second = s2.ToCharArray();
            if (first.Length == 0)
            {
                return second.Length;
            }
            if (second.Length == 0)
            {
                return first.Length;
            }
            int[,] d = new int[first.Length + 1, second.Length + 1];
            for (int i = 0; i <= first.Length; i++)
            {
                d[i, 0] = i;
            }
            for (int j = 0; j <= second.Length; j++)
            {
```

```

        d[0, j] = j;
    }
    for (int i = 1; i <= first.Length; i++)
    {
        for (int j = 1; j <= second.Length; j++)
        {
            int cost = (second[j - 1] == first[i - 1]) ? 0 : 1;
            d[i, j] = Minimum(
                d[i - 1, j] + 1,
                d[i, j - 1] + 1,
                d[i - 1, j - 1] + cost
            );
        }
    }
    return d[first.Length, second.Length];
}

static void DisplayResults(string s1, string s2, int d)
{
    Console.WriteLine("distance between {0} and {1}: {2}",
        s1, s2, d);
}

static void Main(string[] args)
{
    int d = 0;
    string s1 = "";
    string s2 = "";

    s1 = "";
    s2 = "aunt";
    d = ComputeTheDistanceBetween(s1, s2);
    DisplayResults(s1, s2, d);

    s1 = "kitten";
    s2 = "";
    d = ComputeTheDistanceBetween(s1, s2);
    DisplayResults(s1, s2, d);

    s1 = "ant";

```

```

        s2 = "aunt";
        d = ComputeTheDistanceBetween(s1, s2);
        DisplayResults(s1, s2, d);

        s1 = "kitten";
        s2 = "sitting";
        d = ComputeTheDistanceBetween(s1, s2);
        DisplayResults(s1, s2, d);
        Console.WriteLine("Press enter/return to exit");
        Console.ReadLine();
    }
}
}

```

Answer:



```

選取 C:\Program Files\dotnet\d...
distance between and aunt: 4
distance between kitten and : 6
distance between ant and aunt: 1
distance between kitten and sitting: 3
Press enter/return to exit

```

7. 時間序列(time series)是一組按照時間發生先後順序進行排列的數據點序列，常用於財務金融、經濟分析、醫學公衛、企業銷售、社會政治、電機資訊的訊號分析等等(所以學經濟的台大管中閔校長，也可能在國際電機電子工程師學會的期刊發表論文)。時間序列隨時間的改變，既有長時間的趨勢，也有短時間或者週期性的變化。尋求長期趨勢的方法之一，稱為移動平均(moving average)。以下用圖 2 所示，美國佛羅里達州新冠肺炎(COVID-19)於 2020 年 3 月到 7 月的每日死亡人數統計，說明移動平均概念(參看 <https://statisticsbyjim.com/time-series/moving-averages-smoothing/>)。圖 2 的數據顯示死亡率每天都有快速的變動，比較不容易找出長期變化的趨勢。要除去這些短期改變，最直接的想法，便是選擇一個適當的期間(稱為「窗框」(window)，這裡令為 7 天)，計算這一段時間的平均值，把短期變化消除；接著將窗框移動一天，重新計算平均，再移動一天，再算一次平均，以此類推，可得到如圖 2 中的紅色曲線。COVID-19 死亡率先上升、拉平、下降、再爬升的趨勢就很容易觀察到。這種方法，也稱為平滑化(smoothing)方法。

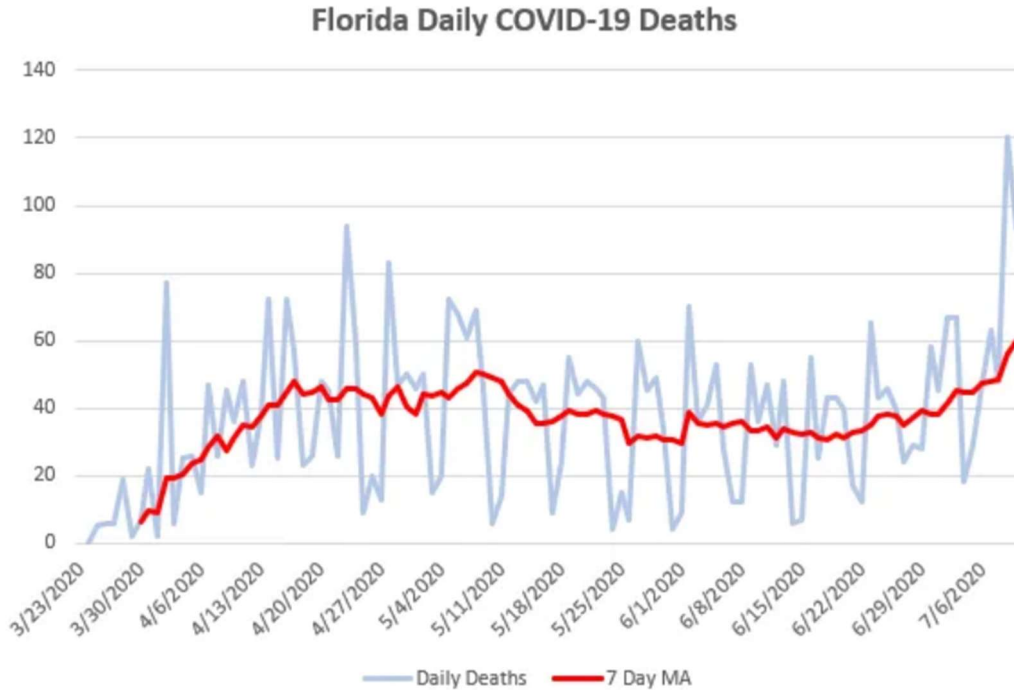


圖2. 美國佛羅里達州新冠肺炎(COVID-19)於2020年3月到7月的每日死亡人數統計
 取自 <https://statisticsbyjim.com/time-series/moving-averages-smoothing/>

基本上，計算移動平均有兩種方法，第一種將窗框平均值，放在窗框最後一點對應的時間點，稱為單邊移動平均(one-sided moving average)，亦即 $MV_t = (X_{t-6} + X_{t-5} + X_{t-4} + X_{t-3} + X_{t-2} + X_{t-1} + X_t)/7$ ，其中 X_t 代表原先時間序列(圖2中的淺藍色線)， MV_t 則是在時間 t 算出的單邊移動平均(圖2中的紅色線)。有了移動平均，可以算出短期變動(variation) $V_t = X_t - MV_t$ 。

第二種方法，則是將窗框平均值放在窗框中心，稱為中心移動平均(centered moving average)。計算方式為 $MV_t = (X_{t-3} + X_{t-2} + X_{t-1} + X_t + X_{t+1} + X_{t+2} + X_{t+3})/7$ 。對應的短期變動也可由 $V_t = X_t - MV_t$ 算出。

這兩種移動平均的方法，都是數位訊號處理(digital signal processing，簡稱DSP)領域中，Finite Impulse Response (簡稱FIR) 數位濾波器(digital filter)的特例，有相關理論可分析其特性。

本題希望你能夠寫一個程式，計算美國佛羅里達州新冠肺炎(COVID-19)於2020年3月22日到4月27日，共36天，每日死亡人數統計的單邊移動平均及中心移動平均，以及對應的短期變動。令窗框大小均為 7。程式執行的螢幕截圖如圖3。


```

*** Florida daily deaths moving average: one-sided***
date: 2020/3/23, daily deaths: 18, moving average: 0, variation: 18
date: 2020/3/24, daily deaths: 18, moving average: 0, variation: 18
date: 2020/3/25, daily deaths: 23, moving average: 0, variation: 23
date: 2020/3/26, daily deaths: 29, moving average: 0, variation: 29
date: 2020/3/27, daily deaths: 35, moving average: 0, variation: 35
date: 2020/3/28, daily deaths: 54, moving average: 0, variation: 54
date: 2020/3/29, daily deaths: 56, moving average: 33.285714285714285, variation: 22.714285714285715
date: 2020/3/30, daily deaths: 63, moving average: 39.714285714285715, variation: 23.285714285714285
date: 2020/3/31, daily deaths: 85, moving average: 49.285714285714285, variation: 35.714285714285715
date: 2020/4/1, daily deaths: 87, moving average: 58.42857142857143, variation: 28.57142857142857
date: 2020/4/2, daily deaths: 164, moving average: 77.71428571428571, variation: 86.28571428571429
date: 2020/4/3, daily deaths: 170, moving average: 97, variation: 73
date: 2020/4/4, daily deaths: 195, moving average: 117.14285714285714, variation: 77.85714285714286
date: 2020/4/5, daily deaths: 221, moving average: 140.71428571428572, variation: 80.28571428571428
date: 2020/4/6, daily deaths: 236, moving average: 165.42857142857142, variation: 70.57142857142858
date: 2020/4/7, daily deaths: 283, moving average: 193.71428571428572, variation: 89.28571428571428
date: 2020/4/8, daily deaths: 309, moving average: 225.42857142857142, variation: 83.57142857142858
date: 2020/4/9, daily deaths: 354, moving average: 252.57142857142858, variation: 101.42857142857142
date: 2020/4/10, daily deaths: 390, moving average: 284, variation: 106
date: 2020/4/11, daily deaths: 438, moving average: 318.7142857142857, variation: 119.28571428571428
date: 2020/4/12, daily deaths: 461, moving average: 353, variation: 108
date: 2020/4/13, daily deaths: 499, moving average: 390.57142857142856, variation: 108.42857142857144
date: 2020/4/14, daily deaths: 571, moving average: 431.7142857142857, variation: 139.28571428571428
date: 2020/4/15, daily deaths: 596, moving average: 472.7142857142857, variation: 123.28571428571428
date: 2020/4/16, daily deaths: 668, moving average: 517.5714285714286, variation: 150.42857142857144
date: 2020/4/17, daily deaths: 725, moving average: 565.4285714285714, variation: 159.57142857142856
date: 2020/4/18, daily deaths: 748, moving average: 609.7142857142857, variation: 138.28571428571433
date: 2020/4/19, daily deaths: 774, moving average: 654.4285714285714, variation: 119.57142857142856
date: 2020/4/20, daily deaths: 822, moving average: 700.5714285714286, variation: 121.42857142857144
date: 2020/4/21, daily deaths: 867, moving average: 742.8571428571429, variation: 124.14285714285711
date: 2020/4/22, daily deaths: 893, moving average: 785.2857142857143, variation: 107.71428571428567
date: 2020/4/23, daily deaths: 987, moving average: 830.8571428571429, variation: 156.1428571428571
date: 2020/4/24, daily deaths: 1046, moving average: 876.7142857142857, variation: 169.28571428571433
date: 2020/4/25, daily deaths: 1055, moving average: 920.5714285714286, variation: 134.42857142857144
date: 2020/4/26, daily deaths: 1075, moving average: 963.5714285714286, variation: 111.42857142857144

*** Florida daily deaths moving average: centered***
date: 2020/3/23, daily deaths: 18, moving average: 0, variation: 18
date: 2020/3/24, daily deaths: 18, moving average: 0, variation: 18
date: 2020/3/25, daily deaths: 23, moving average: 0, variation: 23
date: 2020/3/26, daily deaths: 29, moving average: 30.714285714285715, variation: -1.7142857142857153
date: 2020/3/27, daily deaths: 35, moving average: 37.142857142857146, variation: -2.142857142857146
date: 2020/3/28, daily deaths: 54, moving average: 46, variation: 8
date: 2020/3/29, daily deaths: 56, moving average: 54.285714285714285, variation: 1.7142857142857153
date: 2020/3/30, daily deaths: 63, moving average: 72.71428571428571, variation: -9.714285714285708
date: 2020/3/31, daily deaths: 85, moving average: 89.28571428571429, variation: -4.285714285714292
date: 2020/4/1, daily deaths: 87, moving average: 109.14285714285714, variation: -22.14285714285714
date: 2020/4/2, daily deaths: 164, moving average: 131.71428571428572, variation: 32.28571428571428
date: 2020/4/3, daily deaths: 170, moving average: 153.28571428571428, variation: 16.714285714285722
date: 2020/4/4, daily deaths: 195, moving average: 181.28571428571428, variation: 13.714285714285722
date: 2020/4/5, daily deaths: 221, moving average: 202, variation: 19
date: 2020/4/6, daily deaths: 236, moving average: 228.28571428571428, variation: 7.714285714285722
date: 2020/4/7, daily deaths: 283, moving average: 256.14285714285717, variation: 26.857142857142833
date: 2020/4/8, daily deaths: 309, moving average: 287.14285714285717, variation: 21.857142857142833
date: 2020/4/9, daily deaths: 354, moving average: 319.2857142857143, variation: 34.71428571428572
date: 2020/4/10, daily deaths: 390, moving average: 350.14285714285717, variation: 39.85714285714283
date: 2020/4/11, daily deaths: 438, moving average: 387.57142857142856, variation: 50.428571428571445
date: 2020/4/12, daily deaths: 461, moving average: 422.14285714285717, variation: 38.85714285714283
date: 2020/4/13, daily deaths: 499, moving average: 461.85714285714283, variation: 37.14285714285717
date: 2020/4/14, daily deaths: 571, moving average: 502.85714285714283, variation: 68.14285714285717
date: 2020/4/15, daily deaths: 596, moving average: 543.8571428571429, variation: 52.14285714285711
date: 2020/4/16, daily deaths: 668, moving average: 583.1428571428571, variation: 84.85714285714289
date: 2020/4/17, daily deaths: 725, moving average: 619, variation: 106
date: 2020/4/18, daily deaths: 748, moving average: 657.7142857142857, variation: 90.28571428571433
date: 2020/4/19, daily deaths: 774, moving average: 689.8571428571429, variation: 84.14285714285711
date: 2020/4/20, daily deaths: 822, moving average: 727.2857142857143, variation: 94.71428571428567
date: 2020/4/21, daily deaths: 867, moving average: 769.8571428571429, variation: 97.14285714285711
date: 2020/4/22, daily deaths: 893, moving average: 810, variation: 83
date: 2020/4/23, daily deaths: 987, moving average: 846.1428571428571, variation: 140.8571428571429
date: 2020/4/24, daily deaths: 1046, moving average: 877.7142857142857, variation: 168.28571428571433
date: 2020/4/25, daily deaths: 1055, moving average: 0, variation: 1055
date: 2020/4/26, daily deaths: 1075, moving average: 0, variation: 1075
date: 2020/4/27, daily deaths: 1088, moving average: 0, variation: 1088
Press enter/return to exit

```

圖3. 佛羅里達州 COVID-19 於 3/22/2020 至 4/27/2020 的單邊移動平均及中心移動平均計算結果螢幕截圖。

本題所需的數據資料，假定已寫在主程式中如下，可以直接應用：

```

const int N_DAYS = 36;
string[] date = {
    "2020/3/23", "2020/3/24", "2020/3/25", "2020/3/26",
    "2020/3/27", "2020/3/28", "2020/3/29", "2020/3/30",
    "2020/3/31", "2020/4/1", "2020/4/2", "2020/4/3",
    "2020/4/4", "2020/4/5", "2020/4/6", "2020/4/7",
    "2020/4/8", "2020/4/9", "2020/4/10", "2020/4/11",
    "2020/4/12", "2020/4/13", "2020/4/14", "2020/4/15",

```

```

    "2020/4/16", "2020/4/17", "2020/4/18", "2020/4/19",
    "2020/4/20", "2020/4/21", "2020/4/22", "2020/4/23",
    "2020/4/24", "2020/4/25", "2020/4/26", "2020/4/27"
};

```

```

int[] dailyDeaths = {
    18,      18,      23,      29,
    35,      54,      56,      63,
    85,      87,      164,     170,
    195,     221,     236,     283,
    309,     354,     390,     438,
    461,     499,     571,     596,
    668,     725,     748,     774,
    822,     867,     893,     987,
    1046,    1055,    1075,    1088
};

```

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%)

Answer:

```

// One-sided and centered moving average computation
// for Florida daily deaths data
// Reference:
// https://statisticsbyjim.com/time-series/moving-averages-smoothing/

using System;
namespace Problem7
{
    class Program
    {
        static void InitializeMovingAverages(
            double[] movingAverages)
        {
            for(int t = 0; t < movingAverages.Length; ++t)
            {
                movingAverages[t] = 0.0;
            }
        }

        static void InitializeVariations(double[] variations)
        {
            for(int t = 0; t < variations.Length; ++t)
            {
                variations[t] = 0.0;
            }
        }

        static void ComputeOneSidedMovingAverages(
            int windowSize, int[] dailyDeaths, double[] movingAverages)
        {
            int sum;
            for(int t = windowSize-1; t < dailyDeaths.Length; ++t)
            {
                sum = 0;
                for(int tau = 0; tau < windowSize; ++tau)

```



```

        {
            sum += dailyDeaths[t-tau];
        }
        movingAverages[t] = (double) sum / windowSize;
    }
}

static void ComputeCenteredMovingAverages(
    int windowSize, int[] dailyDeaths, double[] movingAverages)
{
    int sum;
    for(int t = windowSize/2;
        t < dailyDeaths.Length-windowSize/2; ++t)
    {
        sum = 0;
        for(int tau = -windowSize/2; tau < windowSize/2; ++tau)
        {
            sum += dailyDeaths[t-tau];
        }
        movingAverages[t] = (double) sum / windowSize;
    }
}

static void ComputeVariations(
    int[] dailyDeaths, double[] movingAverages,
    double[] variations)
{
    for(int t = 0; t < dailyDeaths.Length; ++t)
    {
        variations[t] = dailyDeaths[t] - movingAverages[t];
    }
}

static void DisplayBanner(string title)
{
    Console.WriteLine("*** " + title + "***");
}

static void DisplayResults(
    string[] date, int[] dailyDeaths,
    double[] movingAverages, double[] variations)
{
    for(int t = 0; t < dailyDeaths.Length; ++t)
    {
        Console.WriteLine(
            "date: {0}, daily deaths: {1}, moving average: {2}, " +
            "variation: {3}",
            date[t], dailyDeaths[t],
            movingAverages[t], variations[t]);
    }
}

static void Main(string[] args)
{
    const int N_DAYS = 36;
    string[] date = {
        "2020/3/23", "2020/3/24", "2020/3/25", "2020/3/26",
        "2020/3/27", "2020/3/28", "2020/3/29", "2020/3/30",
        "2020/3/31", "2020/4/1", "2020/4/2", "2020/4/3",
        "2020/4/4", "2020/4/5", "2020/4/6", "2020/4/7",
        "2020/4/8", "2020/4/9", "2020/4/10", "2020/4/11",
        "2020/4/12", "2020/4/13", "2020/4/14", "2020/4/15",

```

```

        "2020/4/16", "2020/4/17", "2020/4/18", "2020/4/19",
        "2020/4/20", "2020/4/21", "2020/4/22", "2020/4/23",
        "2020/4/24", "2020/4/25", "2020/4/26", "2020/4/27"
    };

    int[] dailyDeaths = {
        18,      18,      23,      29,
        35,      54,      56,      63,
        85,      87,      164,     170,
        195,     221,     236,     283,
        309,     354,     390,     438,
        461,     499,     571,     596,
        668,     725,     748,     774,
        822,     867,     893,     987,
        1046,    1055,    1075,    1088
    };

    double[] movingAverages = new double[N_DAYS];
    double[] variations = new double[N_DAYS];

    const int WINDOW_SIZE = 7;

    // one-sided moving average
    InitializeMovingAverages(movingAverages);
    InitializeVariations(variations);
    ComputeOneSidedMovingAverages(WINDOW_SIZE, dailyDeaths,
        movingAverages);
    ComputeVariations(dailyDeaths, movingAverages, variations);
    DisplayBanner(
        "Florida daily deaths moving average: one-sided");
    DisplayResults(date, dailyDeaths,
        movingAverages, variations);

    // centered moving average
    InitializeMovingAverages(movingAverages);
    InitializeVariations(variations);
    ComputeCenteredMovingAverages(WINDOW_SIZE, dailyDeaths,
        movingAverages);
    ComputeVariations(dailyDeaths, movingAverages, variations);
    DisplayBanner(
        "Florida daily deaths moving average: centered");
    DisplayResults(date, dailyDeaths,
        movingAverages, variations);

    Console.WriteLine("Press enter/return to exit");
    Console.ReadLine();
}
}
}

```