البرمجة الإجرائية
Procedural Programming

IPG202

الفصل الأول

أنواع البيانات المركبة : المصفوفات

Compound Data Types: Arrays

الكلمات المفتاحية

بنية مركبة، مصفوفة، مصفوفة أحادية، مصفوفة ثنائية، مصفوفة مسننة، وصول مباشر، وصول عشوائي، دليل، فهرس، foreach.

ملخص الفصل

يستعرض هذا الفصل مفهوم بنى المعطيات المركبة، حيث يركز على المصفوفات، فيعرض أسلوب تعريفها في لغة C#، وكيفية تهيئتها بقيم إبتدائية والوصول إلى عناصرها وطريقة تخزينها في ذاكرة الحاسوب. كما يقدم الأمر التكراري foreach الخاص بالتجول على قيم المصفوفة.

أهداف الفصل

بنهاية هذا الفصل سيكون الطالب قادراً على:

- معرفة طريقة تحقيق المصفوفات بلغة C#.
- التصريح عن المصفوفة.
- التعامل مع المصفوفات، إخراج وإدخال البيانات منها وإليها .
- استخدام المعامل الدليل في الوصول إلى عناصر المصفوفة.
- التعامل مع المصفوفة المسننة.

محتويات الفصل

1. مقدمة
2. ماهي المصفوفات وماهي أنواعها.
3. المصفوفة أحادية البعد.
4. أمثلة حول استخدام المصفوفات أحادية البعد.
5. المصفوفة متعددة الأبعاد.
6. أمثلة حول استخدام المصفوفات ثنائية البعد.
7. تمارين وأنشطة.

1- مقدمة.

رأينا في الفصل الخامس أنه يمكن تصنيف أنواع البيانات في لغة C# من حيث التركيب إلى بيانات بسيطة simple or fundamental وبيانات مركبة compound.

كما تعرفنا إلى البيانات البسيطة ورأينا أن من أهم صفاتها أن المتحول من نوع البيانات البسيطة قادر على تخزين قيمة واحدة في نفس الوقت.

بما أن على البرمجة أن تعكس حلولاً لمسائل العالم الحقيقي، ونمذجة لهذه الحلول باستخدام الحاسوب. فإن الأنواع البسيطة لوحدها تبقي لغات البرمجة قاصرة عن التعامل مع طيف واسع من المسائل، إذ تتضمن العديد من المسائل معالجة مجاميع كبيرة من البيانات المترابطة ببعضها والتي يمكن نمذجتها رياضياً باستخدام السلاسل والمتتاليات.

يتطلب نمذجة مثل هذه المجاميع أنواع بيانات قادرة على تخزين كم كبير من القيم في الوقت نفسه، لهذه الغاية قدمت لغة C# أنواع البيانات المركبة compound.

يمكن للقيم في مجاميع البيانات أن تكون من النوع نفسه، ولهذا الغرض تضمنت لغة C# بنى مسبقة التعريف تدعى مصفوفات القيم الرقمية –بحالتها العامة– arrays، ومصفوفات القيم المحرفية –بحالتها الخاصة– strings.

2- ماهي المصفوفات، وما هي أنواعها

يمكن القول بأن **المصفوفة array** – من وجهة نظر البيانات – **هي مجموعة من البيانات أو تتالي من عناصر البيانات ذو حجم ثابت (مجموعة مرقمة) وجميعها من نفس النوع.**
أما من وجهة نظر الذاكرة، **فالمصفوفة هي عبارة عن مجموعة من خانات الذاكرة المتتالية والمتجاورة والتي تملك نفس الاسم ونفس النوع.**

وبالتالي فإن للمصفوفة عدداً محدداً من العناصر، وهي مرقمة، وبالتالي فهناك عنصر أول، عنصر ثاني، .... إلخ. كما أن عناصر المصفوفة يجب أن تكون من النوع نفسه (أي مصفوفة من الأعداد الصحيحة، مصفوفة من الأعداد الحقيقية، مصفوفة من المصفوفات ...). وبالتالي فإن المصفوفة تعتبر مناسبة جداً لتنظيم مجموعات البيانات المتجانسة.

تتميز المصفوفة بإمكانية **الوصول المباشر direct access أو العشوائي random access** إلى أي من قيمها، بمعنى آخر، إن الوقت اللازم للوصول إلى أي عنصر ضمن المصفوفة هو نفسه بغض النظر عن موقعه ضمن المصفوفة.

بما أن العناصر مرقمة، فإن عملية تحديد موقع العنصر ضمن المصفوفة تتم من خلال رقمه، (دليله أو فهرسه) index or subscript.

يمكن أن يتم تحديد موقع العنصر في المصفوفة من خلال دليل واحد وعندها نقول بأن هذه المصفوفة أحادية البعد one-dimensional، ويحتاج في بعض الأحيان إلى أكثر من دليل وعندها نقول بأن هذه المصفوفة متعددة الأبعاد multi-dimensional.

3- المصفوفة أحادية البعد

كما أصبح واضحاً، فالمصفوفة أحادية البعد (وتدعى أحياناً بالنسق) هي عبارة عن تتابع من القيم من نفس النوع والتي يتطلب الوصول إلى أي قيمة منها تحديد دليل واحد إلى موقعه، حيث ترقم المواقع بدءً من الصفر.

التصريح عن المصفوفة

يمكن أن يتم التصريح عن المصفوفة أحادية البعد في لغة C# بثلاث طرق:

الطريقة الأولى: التصريح عن المصفوفة دون تحديد حجمها ويأخذ هذا التصريح الشكل العام التالي:

type [ ] array_name;

حيث:

type هو نوع عناصر المصفوفة ويمكن أن يكون من أي نوع معرف.

array_name هو اسم المصفوفة التي يتم تعريفها.

أمثلة:

```
int [ ] table1;   // Table of integre
char [ ] table2; // Table of char
float [ ] table3;// Table of float
string [ ] tableStr;// Table of string
```

ملاحظة: لا يقوم الأمر السابق بإنشاء عناصر المصفوفة وحجز مكان لها في الذاكرة.

الطريقة الثانية: التصريح عن المصفوفة مع تحديد حجمها ويأخذ هذا التصريح الشكل العام التالي:

type [ ] array_name = new type [size] ;

حيث:

new كلمة مفتاحية تؤدي لحجز عناصر المصفوفة في الذاكرة.

Size هو الحجم الأعظمي للمصفوفة.

أمثلة:

```
int [ ] table1 = new int [5];
char [ ] table2  = new char [12];
float [ ] table3  = new float [8];
string [ ] tableStr  = new String [9];
```

ملاحظة: يقوم الأمر السابق بإنشاء عناصر المصفوفة وحجز مكان لها في الذاكرة وجعل جميع قيم عناصرها أصفاراً.

الطريقة الثالثة: التصريح عن المصفوفة مع إعطائها قيماً أولية ويأخذ هذا التصريح الشكل العام التالي:

type [ ] array_name = { list of initial values } ;

حيث:

list of initial values مجموعة العناصر الأولية للمصفوفة وتكون مفصولة فيما بينها بفواصل.

أمثلة:

```
int [ ] table1 = {17,-9,4,3,57};
char [ ] table2  = {'a','j','k','m','z'};
float [ ] table3  = {-15.7f, 75, -22.03f, 3 ,57 };
string [ ] tableStr  = {"cat","dog","mouse","cow"};
```

ملاحظات:

- يقوم الأمر السابق بإنشاء عناصر المصفوفة وحجز مكان لها في الذاكرة وتخزين القيم الممررة في هذه الأماكن.
- يكون الحجم الأعظمي للمصفوفة في هذه الحالة هو نفس عدد القيم الممررة.

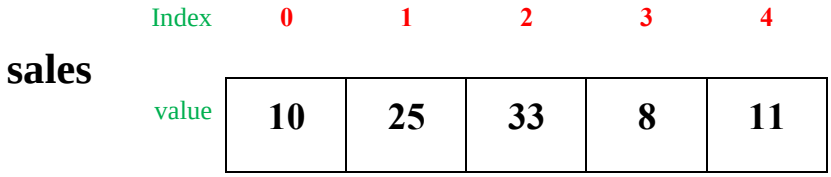
الوصول إلى عناصر المصفوفة

عندما نقوم بإنشاء مصفوفة ما في لغة C#، فإن مترجم اللغة يقوم بحجز أماكن في الذاكرة مخصصة لتخزين عناصر هذه المصفوفة، كما يقوم بترقيم مواقع هذه العناصر اعتباراً من الصفر تسمى أرقام عناصر المصفوفة باسم الفهارس indexs.

فمثلاً، عند التصريح عن المصفوفة sales كما يلي:

```
int[] sales = { 10, 25, 33, 8, 11 };
```

فإن المترجم يقوم بحجز 5 أماكن في الذاكرة لتخزين عناصر هذه العناصر وترقيم هذه المواقع اعتباراً من 0 وحتى 4 كما يبين الشكل التوضيحي التالي:



وبالتالي للوصول إلى أي عنصر، لابد من معرفة رقم فهرسه أو دليله، كمايلي:

```
int[] sales = { 10, 25, 33, 8, 11 };

// طباعة قيمة العنصر الثالث في المصفوفة
Console.WriteLine(sales[2]);

//إسناد قيمة للعنصر الأول في المصفوفة
 sales[0] = 500;

//استخدام عناصر المصفوفة في العمليات والتعابير الحسابية
sales[4] = sales[1] * 4;
int sum_sales = sales[0] + sales[1] + sales[2] + sales[3] + sales[4];
```

من أكثر الأخطاء شيوعاً لدى المبرمجين عند التعامل مع المصفوفات، هو محاولة الوصول إلى عنصر خارج حدود المصفوفة، فمثلاً سيؤدي محاولة تنفيذ الأمر التالي:

```
sales[10] = 500;
```

إلى إطلاق استثناء IndexOutOfRangeException.

التجول عبر المصفوفة

تتميز المصفوفات بخاصية الوصول المباشر direct access أو العشوائي random access ويقصد به أنه يمكن الوصول إلى أي عنصر في المصفوفة بشكل مباشر دون المرور بباقي العناصر.

إن ميزة الوصول العشوائي ملائمة تماماً عندما نرغب بإجراء معالجة ما (إدخال، إسناد، استخدام، مقارنة ... إلخ) على قيمة عنصر محدد في المصفوفة، أما إن كانت رغبتنا هي إجراء معالجة ما لقيم المصفوفة ككل (أو مجموعة جزئية منها) يصبح استخدام ميزة الوصول العشوائي أو المباشر أمراً غير مجد من الناحية البرمجية.

على سبيل المثال، إذا أردنا طباعة قيم جميع عناصر المصفوفة sales باستخدام ميزة الوصول العشوائي، فإننا سنكتب كمايلي:

```
int[] sales = { 10, 25, 33, 8, 11 };
Console.WriteLine(sales[0]);
Console.WriteLine(sales[1]);
Console.WriteLine(sales[2]);
Console.WriteLine(sales[3]);
Console.WriteLine(sales[4]);
```

لا يمكن اعتبار الممارسة السابقة مجدية، فماذا لو كان عدد عناصر المصفوفة 10000 عنصر، فهل سنكتب 10000 أمر طباعة ضمن البرنامج؟!!!

إن الممارسة الأفضل هنا، هي التجول (أي المرور) على عناصر المصفوفة عنصراً عنصراً باستخدام حلقة تكرارية ومن ثم إجراء المعالجة اللازمة في كل تكرار.
يوضح المقطع التالي، كيفية التجول على عناصر المصفوفة sales باستخدام الحلقة التكرارية for:

```
int[] sales = { 10, 25, 33, 8, 11 };
for (int i =0; i< 5; i++)
        Console.WriteLine(sales[i]);
```

نلاحظ في المقطع السابق أننا قمنا بتمرير المتحول i على جميع فهارس المصفوفة، ابتداء من أول عنصر (ذي الفهرس 0) وانتهاء بآخر عنصر (أي ذي الفهرس 4).

حسناً ماذا لو يكن بإمكاننا معرفة عدد عناصر المصفوفة بشكل مسبق؟

تقدم لنا لغة C# حلان لهذا الأمر:

الحل الأول: استخدام التابع length الذي يمكننا من معرفة عدد عناصر المصفوفة في أي لحظة، ومن الممكن استخدامه ضمن الحلقة التكرارية كما يلي:

```
int[] sales = { 10, 25, 33, 8, 11 };
for (int i =0; i< sales.length; i++)
        Console.WriteLine(sales[i]);
```

الحل الثاني: استخدام الأمر التكراري foreach الخاص بالتعامل مع المصفوفات في لغة #C كما يلي:

```
int[] sales = { 10, 25, 33, 8, 11 };
foreach (int i  in sales)
        Console.WriteLine(i);
```

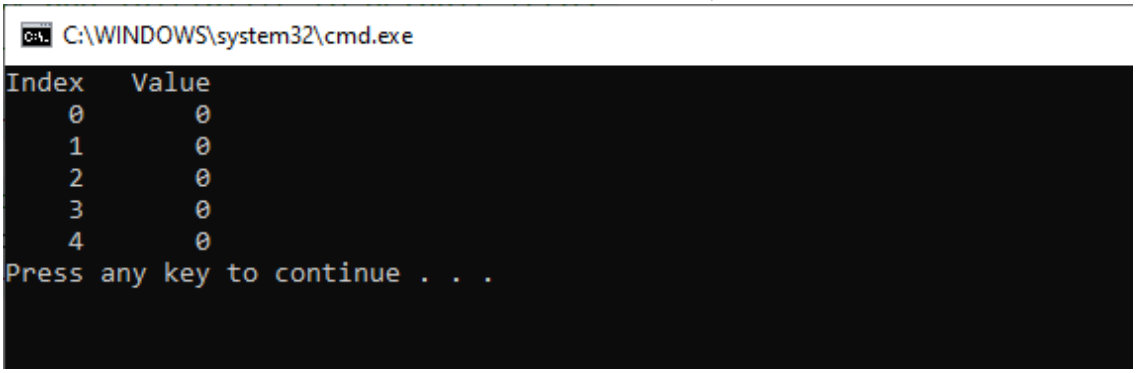
حيث يسمح الأمر foreach بالمرور على عناصر المصفوفة دون الحاجة لمعرف فهارسها أو حجمها.

4- أمثلة حول استخدام المصفوفات أحادية البعد

مثال 1- التحقق من أن عناصر المصفوفة تأخذ القيمة 0 عند إنشائها بدون تمهيدها بقيم ابتدائية

```
int[] array;
array = new int[ 5 ];
Console.WriteLine( "{0}{1,8}", "Index", "Value" );
for ( int counter = 0; counter < array.Length; ++counter )
        Console.WriteLine( "{0,5}{1,8}", counter, array[ counter ] );
```

بتنفيذ هذا المقطع تحصل على الخرج التالي:



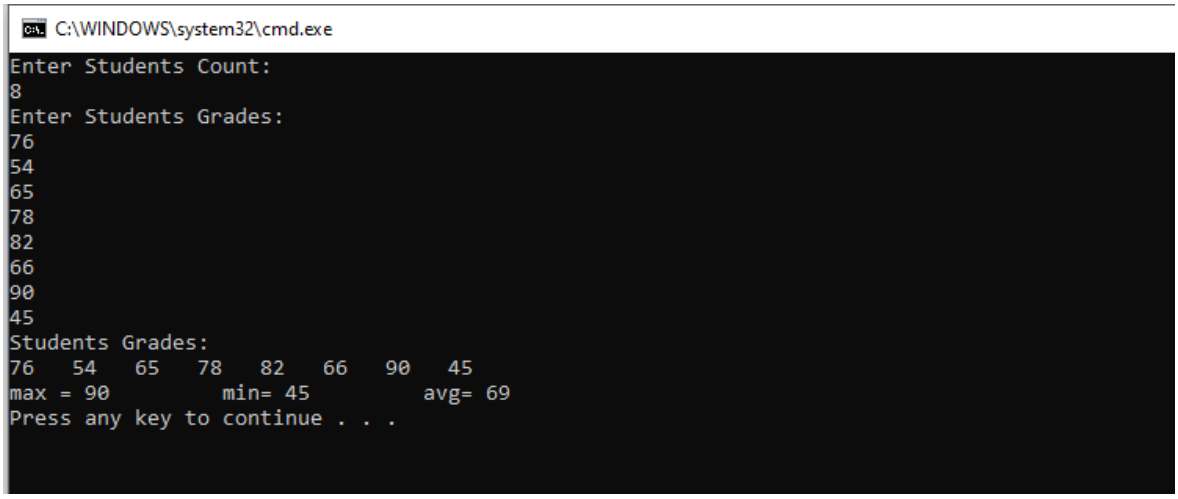
تم إنشاء مصفوفة ذات 5 عناصر، وتهيئتها بالقيمة 0 إفتراضياً.

مثال 2- إدخال علاملات الطلاب في مقرر مقدمة إلى البرمجة ومن ثم إيجاد أدنى علامة وأعلى علامة والمتوسط الحسابي لعلامات الصف.

```
int n, max, min, sum, avg;
Console.WriteLine("Enter Students Count:");
n = Int32.Parse(Console.ReadLine());

//dynamically define students count
int[] grades = new int[n];

//read students grades
Console.WriteLine("Enter Students Grades:");
for (int index=0;index < grades.Length; index++)
        grades[index] = Int32.Parse(Console.ReadLine());

//finding max,min,average
 max = min = sum = grades[0];
 for (int index = 1; index < grades.Length; index++)
       {
          if (grades[index] > max)
                max = grades[index];
           if (grades[index] < min)
                min = grades[index];
            sum += grades[index];
         }
avg = sum/n;

 //printing students grades
Console.WriteLine("Students Grades:");
foreach (int grad in grades)
       Console.Write(grad + "   ");

Console.WriteLine();

//printing max,min,average
 Console.WriteLine("max = {0} \t min= {1} \t avg= {2}", max, min, avg);
```

قمنا في هذا المقطع بتعريف عدد عناصرالمصفوفة بشكل ديناميكي، كما قمنا باستخدام أكثر من صيغة للتجول على عناصر المصفوفة وتنفيذ العمليات عليها.

بتنفيذ هذا المقطع تحصل على الخرج التالي:



5- المصفوفات متعددة الأبعاد

تدعم لغة C# إنشاء مصفوفات متعددة الأبعاد (حتى 32 بعداً). يمكن أن يتم التصريح عن المصفوفة متعددة الأبعاد بإضافة فواصل بين الأقواس المربعة، فعلى سبيل المثال [,] تصرح عن مصفوفة ثنائية البعد، [, ,] تصرح عن مصفوفة ثلاثية .... وهكذا.

الصيغة العامة للتصريح عن مصفوفة متعددة الأبعاد

type [, , ….. ,] array_name;

حيث:

type هو نوع عناصر المصفوفة ويمكن أن يكون من أي نوع معرف.

array_name هو اسم المصفوفة التي يتم تعريفها.

أمثلة:

```
int[,] arr2d; // two-dimensional array
int[, ,] arr3d; // three-dimensional array
int[, , ,] arr4d ; // four-dimensional array
int[, , , ,] arr5d; // five-dimensional array
```

ملاحظة: عدد الفواصل يساعد عدد الأبعاد ناقصاً 1.

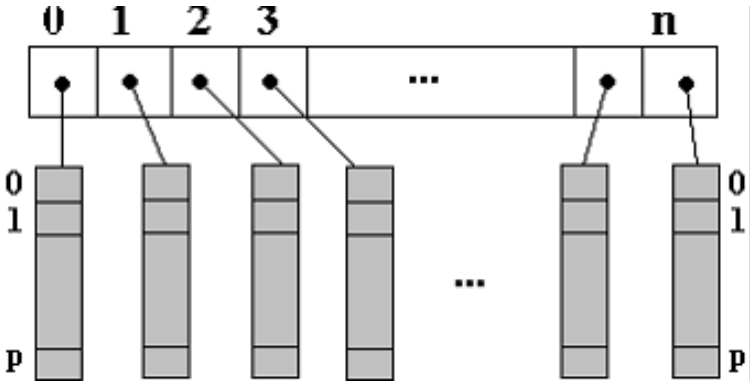
نهتم في دراستنا الحالية فقط بالمصفوفة ثنائية البعد.

المصفوفة ثنائية البعد

يكون لهذه المصفوفة بعدان فقط، ويمكن تمثيلها على شكل جدول، حيث يمثل البعد الأول عدد الأسطر rows والبعد الثاني عدد الأعمدة columns، ويكون للمصفوفة في هذه الحالة فهرسان (فهرس للأسطر وفهرس للأعمدة).

يمكن النظر إلى المصفوفة ثنائية البعد على أنها مصفوفة أحادية البعد، كل عنصر من عناصرها هو عبارة عن مصفوفة أحادية البعد.

يبين الشكل (1) كيفية تمثيل مصفوفة ثنائية البعد ذات p سطر و n عمود.



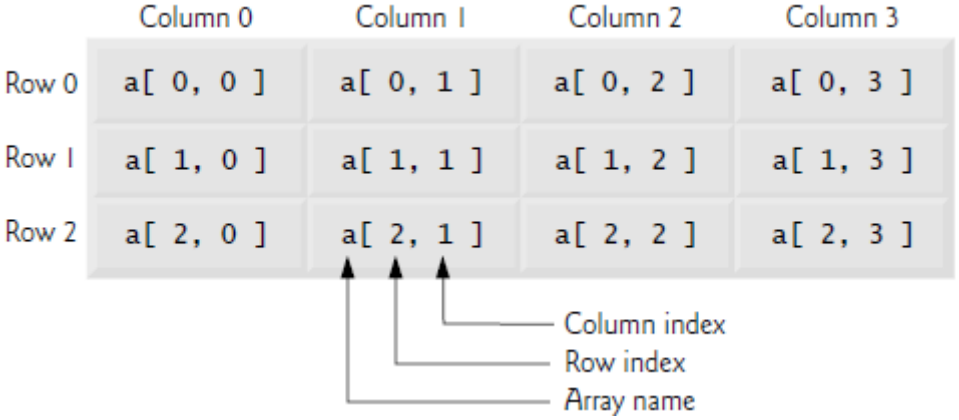
الشكل 1– تمثيل المصفوفة ثنائية البعد

تدعم لغة C# نوعين من المصفوفات الثنائية:

- المصفوفات المستطيلة (أو المربعة) Rectangular or Square
- المصفوفات المسننة Jagged.

المصفوفات المستطيلة

في هذا النوع من المصفوفات، تكون جميع أسطر المصفوفة متساوية الطول (وكذلك جميع الأعمدة)، وبالتالي يمكن تمثيلها على شكل جدول:



الشكل 2– تمثيل مصفوفة مستطيلة ذات 3 أسطر و 4 أعمدة

أمثلة:

```
//declaring and initializing 2-d array
int[,] arr2d = new int[3,2]{
                        {1, 2},
                        {3, 4},
                        {5, 6}
                    };

// or
int[,] arr2d = {
                {1, 2},
                {3, 4},
                {5, 6}
            };
```

يتطلب الوصول إلى أي عنصر في المصفوفة إلى معرفة فهرس السطر وفهرس العمود الذي ينتمي إليه، كما يلي:

```
int[,] arr2d = new int[3,2]{ {1, 2}, {3, 4}, {5, 6} };

arr2d[0, 0]; //returns 1
arr2d[0, 1]; //returns 2
arr2d[1, 0]; //returns 3
arr2d[1, 1]; //returns 4
arr2d[2, 0]; //returns 5
arr2d[2, 1]; //returns 6
//arr2d[3, 0]; //throws run-time error as there is no 4th row
```

في حالة خاصة، إذا تساوى عدد الأسطر وعدد الأعمدة، يقال عن المصفوفة أنها مربعة square.

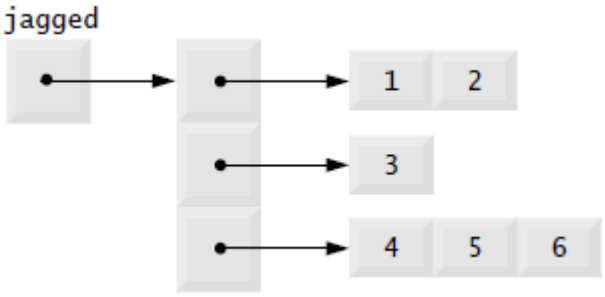
المصفوفات المسننة

في هذا النوع من المصفوفات، تكون أسطر المصفوفة متفاوتة في الطول (أي يكون لكل سطر عدد مختلف من الأعمدة).

كمثال عليها:

```
int[][] jagged = { new int[] { 1, 2 },
new int[] { 3 },
new int[] { 4, 5, 6 } };
```

يمكن تمثيل المصفوفة السابقة على الشكل التالي:



الشكل 3- تمثيل مصفوفة مسننة

في المثال السابق، قمنا بالتصريح عن مصفوفة مسننة وتحديد عدد أسطرها وعدد أعمدة كل صف من خلال عملية التهيئة بقيم أولية.

من الممكن في لغة C# أن نصرح عن المصفوفة المسننة بطريقة أخرى، تتضمن هذه الطريقة تحديد عدد الأسطر مسبقاً ومن ثم إنشاء مصفوفة كل سطر منفردة، كما في المثال التالي:

```
int[][] c;
c = new int[ 2 ][ ]; // create 2 rows
c[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
c[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

6- أمثلة حول استخدام المصفوفة ثنائية البعد

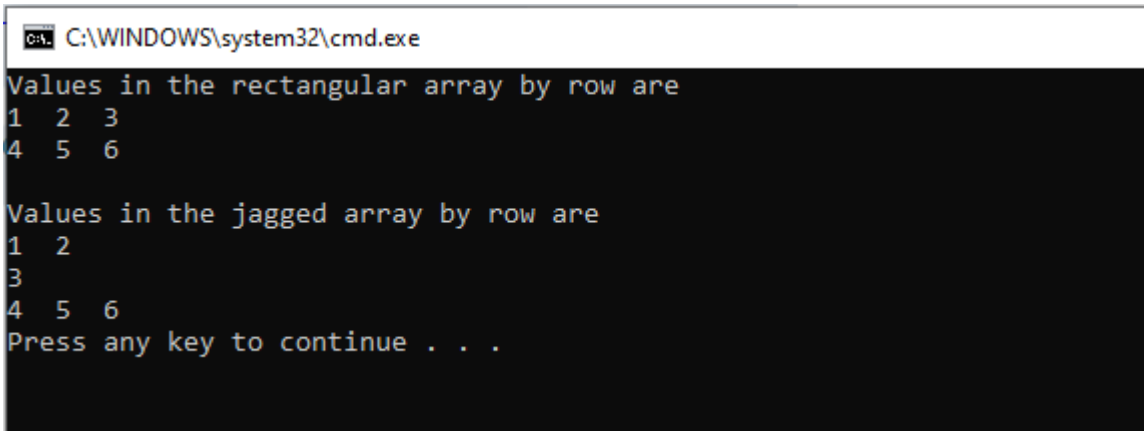
المثال 1

نقوم في المثال التالي بالتصريح عن مصفوفة مستطيلة وعن مصفوفة مسننة وإعطاء قيم ابتدائية لهما ثم نقوم بالتجول عبر هاتين المصفوفتين وطباعة قيمهما (تم استخدام الأمرين for و foreach لتوضيح أسلوب استخدام كل منهما في التجول على المصفوفة الثنائية).

```
// Decalring and Initalizing Rectangular Array
int[,] rectangular = { { 1, 2, 3 }, { 4, 5, 6 } };

// Decalring and Initalizing jagged Array
int[][] jagged = { new int[] { 1, 2 },   new int[] { 3 },   new int[] { 4, 5, 6 } };

// displays array rectangular by row
 Console.WriteLine("Values in the rectangular array by row are");
```

```
// loop through array's rows
for (int row = 0; row < 2; ++row)
{
      // loop through columns of current row
     for (int column = 0; column < 3; ++column)
           Console.Write("{0}  ", rectangular[row, column]);

      Console.WriteLine(); // start new line of output
} // end outer for
Console.WriteLine(); // output a blank line

// displays array jagged by row
Console.WriteLine("Values in the jagged array by row are");

// loop through each row
foreach (int[] row in jagged)
{
       // loop through each element in current row
       foreach (int element in row)
            Console.Write("{0}  ", element);

        Console.WriteLine(); // start new line of output
} // end outer foreach
```

يكون ناتج التنفيذ كما يلي:



المثال 2

نرغب باستخدام المصفوفات في بناء جدول علامات الطلاب، يتضمن الجدول اسماء الطلاب وعلاماتهم في امتحانات 3 مواد ومعدل علامات كل طالب.

للحل سنستخدم مصفوفتان: المصفوفة students لتخزين أسماء الطلاب، المصفوفة grades لتخزين العلامات والمعدلات، وسنقوم بالمزامنة بينهما.

```
//Declaring arrays
string [] students = new string[5];
double [,] grades = new double[5, 4];

double sum;

//reading data and calculating averages
for (int i = 0; i < 5; i++)
{
    Console.Write("ENTER STUDENT No.{0} NAME:", i+1);
    students[i] = Console.ReadLine();
     sum = 0;
    Console.WriteLine("ENTERING GRADES FOR STUDENT No.{0}", i + 1);
    for(int j=0; j<3; j++)
     {
          grades[i, j] = double.Parse(Console.ReadLine());
          sum += grades[i,j];
      }
     grades[i, 3] = sum / 4;
}

//prinintig grades book
Console.WriteLine("CLASS GRAD BOOK");
Console.WriteLine("Name\tMark1\tMark2\tMark3\tAverage");
for (int i = 0; i < 5; i++)
{
    Console.Write(students[i]);
     for (int j = 0; j < 4; j++)
         Console.Write(grades[i, j]+"\t");
         Console.WriteLine();
 }
```

7- تمارين وأنشطة

التمرين الأول:

طور البرنامج المكتوب في المثال رقم 2 من الفقرة السابقة بحيث يقوم بإضافة حقل النتيجة لكل طالب بحيث يكون كل شخص معدلة أقل من 60 راسباً وإلا فإنه ناجح.

التمرين الثاني:

طور البرنامج المكتوب في المثال رقم 2 من الفقرة السابقة بحيث يقوم بحساب نسبة النجاح لإجمالي طلاب الصف.

التمرين الثالث:

أكتب برنامجاً يسمح بإدخال عناصر مصفوفة أحادية البعد مؤلفة من 10 عنصر، ومن ثم حشر عنصر في مكان ما.

ملاحظة: يتم إدخال القيمة المراد حشرها وموقع الحشر من قبل المستخدم.

التمرين الرابع:

نرغب بالقيام بمقارنة مصفوفتين وفق الآلية التالية.

بناء مصفوفة ناتج المقارنة بحيث تكون قيمة كل عنصر من الشكل التالي:

- 0 إذا كان العنصران المقابلان متساويان.
- 1 إذا كان الأول أكبر من الثاني.
- 1- إذا كان الأول أصغر من الثاني.

كما يبين المثال التالي:

المصفوفة الأولى

3	5	5
4	2	1
5	0	2
1	2	4

المصفوفة الثانية

3	0	7
2	4	2
7	1	1
1	2	0

مصفوفة المقارنة

0	1	−1
1	−1	−1
−1	−1	1
0	0	1

حيث أن المصفوفات الثلاث يجب أن تكون لها نفس الأبعاد.

أكتب برنامجاً يقوم بإدخال قيمتي مصفوفتين وإيجاد المصفوفة المعبرة عن مقارنتهما وطباعة النتائج.