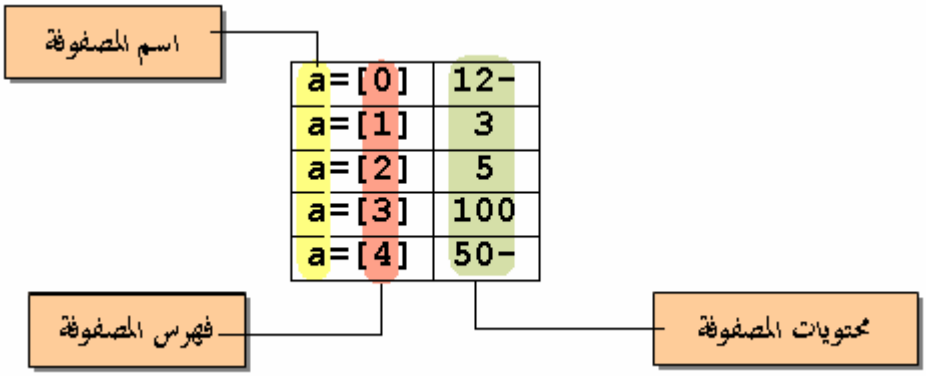
الدرس العاشر : المصفوفات

في هذا الدرس سوف نأخذ أحد أهم بنى البيانات في البرمجة ألا و هي المصفوفات arrays ، و تتميز المصفوفات عن بقية بنى المعطيات الأخرى (مثل القوائم list و الأشجار trees و غيرها) بأن لها حجم ثابت طيلة فترة تنفيذ البرنامج .

و المصفوفة هي عبارة عن مجموعة من خانات الذاكرة المتتالية التي لها نفس الاسم و نفس النوع ، ومن أجل الرجوع إلى خانة معينة من هذه الخانات ضمن المصفوفة يكفي أن نستخدم اسم المصفوفة و رقم موضع الخانة ضمن المصفوفة (الفهرس index).

الشكل التالي يوضح مصفوفة من الأعداد الصحيحة اسمها a و هي تتضمن خمسة عناصر :



من الواضح أن الفهرس يبدأ من الصفر و ليس من الواحد و هذه نقطة مهمة جدا أن تتذكرها عندما تقرر استخدام المصفوفات ، فمثالا العنصر الرابع هو A[3] و هكذا .

و قبل أن ننتقل إلى الجانب العملي للمصفوفات أود أن أجيب على سؤال قد يتبادر إلى ذهن قارئ هذه السطور ألا و هو لماذا المصفوفات ؟!.

لمعرفة أهمية المصفوفات دعني أخبرك أن في الرياضيات فرع بأكمله يبحث في المصفوفات صغيرها و كبيرها ، حيث تسهم المصفوفات في حل الكثير من معضلات العلم الحديث وعلى سبيل المثال يوجد في الهندسة برنامج متخصص في المصفوفات يعرف باسم مختبر المصفوفات Matlab و هو من الاسم متخصص في المصفوفات و كل مهندس تقريبا قد يكون قد سمع عنه (و أنا واحد منهم) مهما كان تخصصه!!.

ثم إن الصور التي تراها ما هي إلا مصفوفات ، فلو لاحظت أنها متكونة من عدة نقاط صغيرة جدا ، و التي تمثل بعناصر المصفوفات .

و لا تنس أن المصفوفات تقوم بدور كبير في تنظيم البرنامج فبدلا أن تعلن عن ألف طالب و هم طلاب مدرسة معينة ، يمكنك أن تقسمهم حسب فصولهم الدراسية في مصفوفات فيتقلص معك عدد المتغيرات إلى 50 متغير (وهي عدد الفصول) و هكذا قس على الباقي .

أرجو الآن قد استوعبت أهمية المصفوفات في عالم البرمجة .

و الآن ننتقل إلى لغة السي شارب .

تعتبر المصفوفات في السي شارب صنف `class` ذا مواصفات معينة تسمى `System.Arry` ، وهي بهذا تختلف عن لغة السي و سي ++ و فيجول بيسك في طريقة تعاملها مع المصفوفات ، و سوف نأخذ طريقة تعاملها بالتفصيل .

أولا : إعلان و إنشاء المصفوفات

بما أن المصفوفات من النوع المرجعي فهي تأخذ خطوتين قبل استخدامها ، الخطوة الأولى: هي الإعلان عنها و في هذه الخطوة نقوم بإنشاء المرجع و تتم في المصفوفة هكذا :

```
اسم المصفوفة [] نوع عناصر المصفوفة
```

مثال :

```
int [] Zayed;
```

للإنشاء مصفوفة باسم `Zayed` تكون عناصرها من نوع `int`

```
string [] B;
```

لتكوين مصفوفة باسم `B` تكون عناصرها سلاسل حرفية .

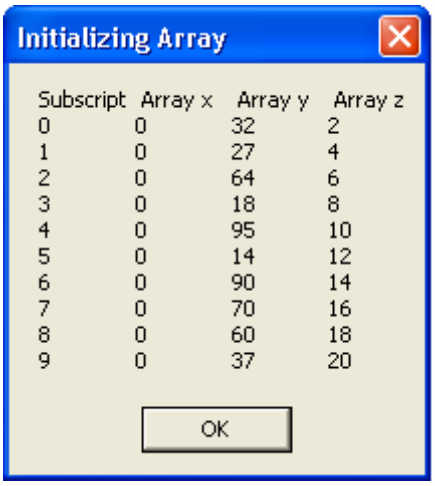
الخطوة الثانية :

وهي تحديد و حجز مقدار معين من الذاكرة للمصفوفة ، بما أن المصفوفة من خصائصها أنها ثابتة الحجم فيجب تحديد عدد عناصر المصفوفة في هذه الخطوة ، و لفعل ذلك يمكنك استخدام رقم أو ثابت أو أن تضع عناصر المصفوفة مباشرة ، المثال التالي يوضح لك كيفية إنشاء مصفوفات بثلاث طرق ، و بعد ذلك يقوم بطباعتهن :

```
using System;
using System.Windows.Forms;
class IntArray
{
      static void Main()
      {
            string output;
             //قمنا بالإعلان عن المصفوفة الأولى
            int [] x;
            // قمنا بحجز الذاكرة لعشرة عناصر بقيمتها الافتراضية
            x = new int[10];
```

```
            //  قمنا بإنشاء المصفوفة الثانية و قمنا بإسناد عناصرها
            // وقيمها مباشرة
            int []y = { 32,27,64,18,95,14,90,70,60,37};
            // أنشأنا ثابت
            const int ARRAY_SIZE = 10;
            // قمنا بإنشاء مصفوفة ثالثة باستخدام الثابت بدل العدد
            int []z = new int[ARRAY_SIZE];
            // قمنا بوضع قيم عناصر المصفوفة الثالثة
            for (int i=0;i< z.Length;i++)
                  z[i] = 2+2*i;

          output = "Subscript \t Array x \t Array y "
                 + "\t Array z \n";
            //  إظهار كل عناصر المصفوفات
            for (int i =0; i < ARRAY_SIZE; i++)
                  output += i + "\t"+ x[i]+ "\t" + y[i] +
                        "\t" + z[i] + "\n";
             MessageBox.Show(output,"Initializing Array");
      }
}
```

الناتج :



من هذا المثال استفدنا أشياء أخرى ، أهمها كيفية إنشاء المصفوفة باستخدام كلمة **new** مع تحديد رقم أو ثابت ، و كيفية إنشاء المصفوفة مباشرة بسرد عناصرها في قوسين {}.

و كذلك يمكنك إنشاء المصفوفة كالتالي :

```
int [] a = new int [3] { 1,2,3};
```

و تعلمنا كيفية إنشاء الثابت بإضافة الكلمة **const** قبل اسم الثابت و نوعه .

و كذلك عرفنا أن هناك خاصية لكل مصفوفة و هي عدد عناصرها و تساوي **ArrayName.length** و كذلك تعلمنا كيفية الوصول إلى عناصر المصفوفة باستخدام الحلقة التكرارية **for** و فهرس المصفوفة .

ثانيا : تمرير المصفوفات إلى الوسائل

قد يكون هذا الموضوع من أكثر المواضيع إرباكا للمتعلمين ، و يكمن لب المشكلة في أن المصفوفات هي نوع مرجعي (أي يحمل مرجع فقط لجسم المصفوفة الحقيقي) و نريد أن نمرر هذا المرجع بواسطة قيمته و مرجعه .

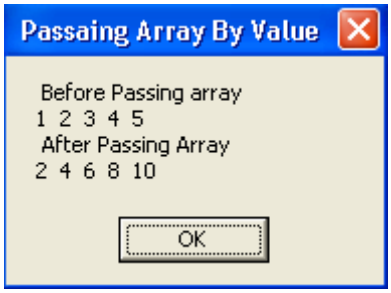
لنأخذ أولا تمرير المصفوفات بواسطة قيمتها :

بهذه الطريقة يمكنك إرسال المصفوفة بمجرد وضع اسم المصفوفة في معاملات الوسيلة ، سوف يتم في أثناء تنفيذ البرنامج إرسال نسخة مرجع المصفوفة إلى الوسيلة ، و بعد ذلك و بواسطة هذا المرجع تستطيع الوسيلة من الاستفادة من المصفوفة و القيام بتغيرها إذا اقتضت الحاجة على ذلك :

خذ المثال التالي :

```
using System;
using System.Windows.Forms;
class PassArrayByValue
{
      static void Main ()
      {
            int []a = {1,2,3,4,5};

            string output = " Before Passing array \n";
            foreach (int i in a)
                  output+= i + "   ";

            ModifyArray(a);// المصفوفة تم تمريرها بواسطة قيمتها

            output +="\n After Passing Array \n";
            foreach ( int i in a)
                  output += i +"   ";

            MessageBox.Show(output ," Passaing Array By Value");
      }

     static void ModifyArray( int []b)
      {
            for ( int j= 0; j<b.Length ;j++)
                  b[j]*=2;
      }
}
```

والنتيجة :



يظهر لك أن تمرير المصفوفة بهذه الطريقة لا يختلف عن تمرير المتغيرات الأخرى ، اللهم إلا في كيفية الإعلان عن المصفوفة في معاملات الوسيلة فقط .

الأمر الثاني : أنه عندما مررنا المصفوفة بطريقة قيمة مرجعها ، قمنا بإرسال مرجع لتلك المصفوفة أو قل جسم المصفوفة الحقيقي ؛ فعلى هذا تمكنا من التغيير في جسم المصفوفة بدون التغيير في المرجع نفسه ، في الجانب الأخر عندما نمرر المصفوفة بواسطة مرجعها سوف نتمكن من تغيير المرجع بنفسه .

لكي تفهم ما معنى أن نغير في المرجع أضف السطر التالي لوسيلة `ModifyArray`

```
b = new int[] {15,17,18,19,20,21};
```

في هذا السطر سوف نحاول أن نغير المرجع `b` إلى مصفوفة جديدة و لكن بدون فائدة ، سوف نكرر هذا السطر في أثناء شرح تمرير المصفوفات عن طريق مراجعها ، و سوف نرى أنه سوف يعمل و يقوم بتغير المرجع .

الأمر الثالث : وهو أن عناصر المصفوفة أثناء تمريرها إلى الوسائل سوف يتعامل معها كأنها متغيرات عادية (أي قيمية) (راجع الدرس التاسع) .

الأمر الرابع : وهو استخدام الحلقة التكرارية الجديدة `foreach` للوصول إلى عناصر المصفوفة ، و هذه الحلقة تستعمل أيضا للمجموعات `collection` و طريقة استخدامها سهلة و هي كالتالي :

```
foreach ( نوع المتغير   المتغير   in   اسم المجموعة )
{
          محتوى الحلقة التكرارية
}
```

حيث يكون نوع المتغير مطابق لنوع عناصر المجموعة ، مثالا إذا كانت عناصر المصفوفة من نوع `int` يجب أن يكون المتغير من ذلك النوع .

و لكن هذه الحلقة هي حلقة للقراءة فقط أي لا تستطيع أن تغير قيم المصفوفة أو المجموعة بشكل عام ، لذا لتغيير محتويات المصفوفة يجب استخدام الحلقة التكرارية `for` .

و الآن لننتقل إلى تمرير المصفوفة بواسطة مرجعها .

تمرير المصفوفات بواسطة المرجع (استخدام `ref`)

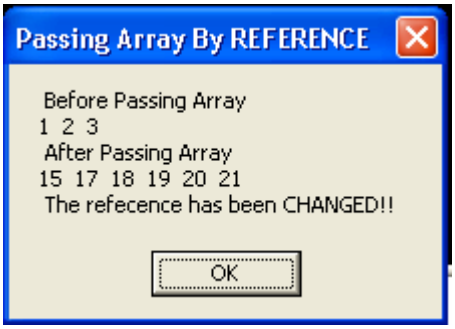
لربما أدركت أهمية تمرير المصفوفات بواسطة قيمة مراجعها بشكل افتراضي (الحالة السابقة) لاختصار الوقت و توفير الذاكرة ، ولكن قد تتساءل إذا كان تمرير الافتراضي للمصفوفات (وغيرها من الأنواع المرجعية) يتم عن طريق قيمة مرجعها فلماذا نستخدم كلمة `ref` لتمرير هذه الأنواع (أي تمرير مراجعها بنفسها و ليس نسخ منها) لماذا هذا التعقيد ؟؟!

و الجواب : لا أدري ، وعندما تجد جوابا لا تبخل به علينا .

على العموم خذ المثال التالي الذي يبين لك إمكانية تغيير المرجع نفسه باستخدام كلمة **ref** :

```
using System;
using System.Windows.Forms;
class PassArrayByRef
{
      static void Main ()
      {
            int[]firstArray ={ 1,2,3}; // تم إنشاء المصفوفة الأولى

            // تم إنشاء نسخة من مرجع المصفوفة الأولى
           int []firstArrayCopy = firstArray;
            // طباعة محتويات المصفوفة قبل تمريرها
            string output=" Before Passing Array \n";
            foreach ( int i in firstArray)
                  output += i +"  ";
            // تمرير المصفوفة بواسطة مرجعها
            ModifyArray(ref firstArray);
            // طباعة محتويات المصفوفة بعد تمريرها
            output +=" \n After Passing Array \n";
            foreach (int i in firstArray)
                  output+= i + "  ";
            // اختبار إذا ما كان مرجع المصفوفة هو نفسه قبل تمريرها
            if ( firstArray == firstArrayCopy)
                  output +="\n The refecence is not CHANGED!!";
            else
                  output+= "\n The refecence has been CHANGED!! ";

            MessageBox.Show(output,"Passing Array By REFERENCE ");
      }
      static void ModifyArray( ref int []b)
      {
            // هنا يتم تغير محتويات المصفوفة
            for ( int j= 0; j<b.Length ;j++)
                  b[j]*=2;
           // هنا يتم تغير المرجع
            b = new int[] {15,17,18,19,20,21};
      }

}
```

والنتيجة :



ثالثا : بعض وسائل المصفوفات

قلنا أن المصفوفات عبارة عن خلية System.Array لها بعض الخصائص و الوسائل ، وقد ذكرنا أحد تلك الخصائص و الآن نذكر بعض الوسائل كالتالي :

1- فرز المصفوفة :

أي ترتيبها وفق نظام معين مثلا تصاعدي أو تنازلي و هذا المجال قامت فيه دراسات واسعة في مجال علوم الحاسوب و نتجت الكثير من الخوارزميات لفرز هذه المصفوفات ، أسهلها الفرز الفقاعي (سوف تجد تفصيله في الواجب ...) على العموم الدوت نت قدمت لنا وسيلة لفرز المصفوفات فرزا تصاعديا و هي sort و تكتب كالتالي :

```
Array.Sort(اسم المصفوفة);
```

2- البحث عن مصفوفة :

و هناك عدة تقنيات منها البحث الخطي و البحث الثنائي وغيرها ، و قد تطرقنا إليها في أسئلة الواجب ، و المهم هنا أنه توجد وسيلة للبحث في المصفوفات تسمى البحث الثنائي و تكتب هكذا :

```
Array.BinarySearch (اسم المصفوفة , القيمة المراد البحث عنها );
```

رابعا : المصفوفات متعددة الأبعاد :

يوجد نوعان من المصفوفات متعددة الأبعاد : مصفوفات متجانسة ، و مصفوفات غير متجانسة و تفصيلها التالي :

1- المصفوفات المتجانسة:

و مثال عليها المصفوفات ثنائية البعد و ثلاثية البعد أو رباعية و هكذا ، فتستطيع بواسطة المصفوفة الثنائية تمثيل جدول ، و بواسطة المصفوفة ثلاثية الأبعاد أيام السنة المقسمة على الشهور .

و كيفية إنشاء مثل هذه المصفوفات لا تختلف كثيرا عن المصفوفة ذات البعد الواحد .

و هذه هي الطريقة للإنشاء مصفوفة ثنائية البعد :

```
int [ , ] a = new int [2 , 2];
a[0,0]=1;
a[0,1]=2;
a[1,0]=3;
a[1,1]=4;
```

و الطريقة المختصرة هي كالتالي :

```
int [,]a = new int[] { {1,2}, {3,4} };
```

أو مباشرا:

```
int [,]b = { {1,2}, {3,4} };
```

و هكذا قس على المصفوفة الثلاثية الأبعاد ، فيمكنك أن تعلنها هكذا :

```
int [, ,] array;
```

و بعد ذلك تقوم بملئها كما سبق .

توجد وسيلة للحصول على طول المصفوفة المتعددة الأبعاد فمثالا إذا أردت أن تحصل على طول البعد الأول من المصفوفة السابقة فعليك أن تكتب التالي :

```
a.GetLength(0);
```

حيث 0 تعبر عن البعد الأول و النتيجة هي 2 .

و الآن لننتقل إلى النوع الثاني .

2- مصفوفة المصفوفات :

لربما أدركت أن النوع السابق من المصفوفات متعددة الأبعاد يستلزم أن يكون لها أبعاد ثابتة ، حيث يجب أن يكون كل بعد له طول ثابت فلذا فهي تسمى المصفوفات المستطيلة لشبه بينها و بين المستطيل .

أما النوع الثاني الذي نحن في صدد بيانه فهو لا يستلزم ذلك ، أي أن طول البعد الواحد غير متجانس، و أقرب تشبيه لها بأنها مصفوفة تحتوي على مصفوفات ، و بالمثال يتضح المقال ، حيث تنشأ هكذا :

```
int [][] c = new int[2][];
      c[0] = new int[] {1,2};
      c[1] = new int[]{3,4,5 };
```

حيث ترى أننا استعملنا الأقواس [] لتحديد عدد المصفوفات التي تحويها ، و نبدأ من اليسار في قراءتها فمثالا :

```
int [][, ,][,]
```

تقرأ مصفوفة وحيدة البعد لمصفوفة ثلاثية الأبعاد لمصفوفة ثنائية البعد من الأعداد الصحيحة ، ولا أعتقد أنك ستحتاج لمثل هذه المصفوفة في الوقت القريب !

على العموم نعود لمثالنا السابق بعد الإعلان عن المصفوفة قمنا بملء المصفوفة بالمصفوفات كلا على حده، حيث أنشأنا المصفوفة الأولى و أسندناها إلى `c[0]` و قمنا بعد ذلك بإنشاء المصفوفة الثانية و إسنادها إلى `c[1]` و هكذا .

يمكن أن تحصل على طول كل مصفوفة بأن تحدد رقمها في المصفوفة الأولى كما في المصفوفة وحيدة البعد مثلا :

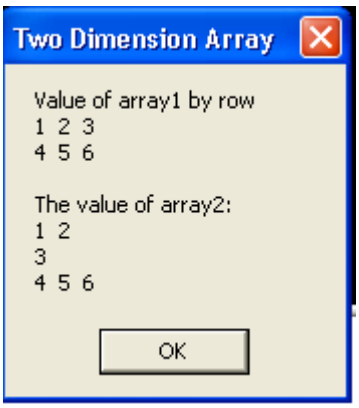
```
c[0].Length;
```

سوف يعطيك عدد عناصر المصفوفة الأولى و هي 2 و `c[1].length` سوف يعطيك 3 .

المثال التالي يوضح الأفكار السابقة :

```
using System;
using System.Windows.Forms;
class MultDim
{
	static void Main ()
	{	// إنشاء مصفوفة ثنائية البعد
		int [,]array1 = new int[,] {{1,2,3,},{4,5,6}};
		// إنشاء مصفوفة المصفوفات
		int [][] array2 = new int[3][];
		array2 [0] = new int[]{1,2};
		array2 [1] = new int[]{3};
		array2 [2] = new int[]{4,5,6};

		// إظهار محتويات المصفوفة الأولى
		string output="Value of array1 by row\n";
		for(int i=0 ;i< array1.GetLength(0);i++)
		{
			for(int j=0; j< array1.GetLength(1); j++)
				output += array1[i,j]+ "  ";
			output +="\n";
		}
		// إظهار محتويات المصفوفة الثانية
		output +="\nThe value of array2:\n";
		for(int i=0; i< array2.Length; i++)
		{
			for(int j=0; j< array2[i].Length; j++)
				output +=array2[i][j] + "  ";
			output +="\n";
		}
		MessageBox.Show(output,"Two Dimension Array ");
	}
}
```

و النتيجة كالتالي:



في المثال السابق استعملنا الحلقة التكرارية **for** ، و لكن ما رأيك أن نستعمل الحلقة التكرارية **foreach** في مصفوفة المصفوفات ؟ قد تتوهم أن استعمال **foreach** سهل ، قد يكون ذلك صحيح بالنسبة للمصفوفات المستطيلة (المتجانسة) و لكن ليس مع المصفوفات غير المتجانسة ، سوف تحتاج إلى حلقتين من **foreach** واحدة تمر على المصفوفات و الأخرى تمر على عناصر تلك المصفوفات .

على سبيل المثال تستطيع أن تكتب الجزء الخاص بالمصفوفة الثانية في المثال السابق هكذا :

```
foreach ( int[] a in array2)
      {
            foreach (int b in a)
                  output += b + "   ";
            output += " \n";
      }
```

قد يبدو هذا أفضل من ذي قبل ، و لكن تذكر أن **foreach** للقراءة فقط .

و الآن خذ المثال التالي :طالب حصل على النتائج التالية في ثلاثة اختبارات في أربع مواد :

	المادة الأولى	المادة الثانية	المادة الثالثة	المادة الرابعة
الاختبار الأول	77	68	86	73
الاختبار الثاني	96	87	89	81
الاختبار الثالث	70	90	86	81

نريد أن نكتب برنامج يقوم بطباعة كشف المواد مع :

1- أعلى و أدني نتيجة في كل الاختبارات .

2- المتوسط الحسابي لكل مادة .

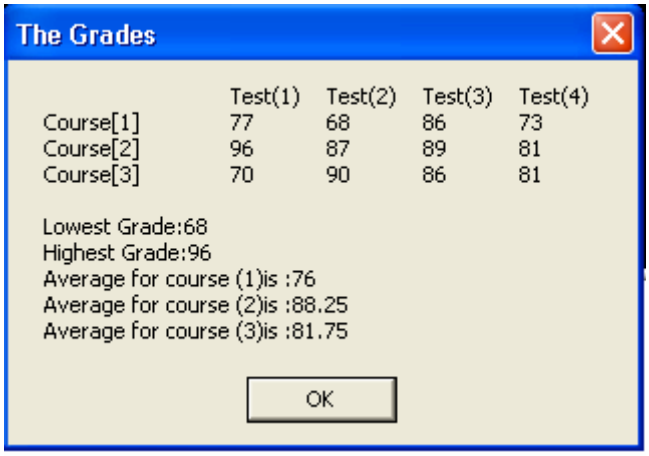
3- أضف إلى ذلك يجب استخدام مصفوفة المصفوفات و **foreach** :

فلنبدأ العمل :

```
using System;
using System.Windows.Forms;
class Grades
{
      static void Main ()
      {      // قمنا بإنشاء المصفوفة التي سوف تحوي الدرجات
            int [][] grades = new int [3][];
            grades[0] = new int[] {77,68,86,73};
            grades[1] = new int[] {96,87,89,81};
            grades[2] = new int[] {70,90,86,81};
            // هنا قمنا بتشكيل الجدول
            string output = "\t\t";
            for(int i=0; i<grades[1].Length ;i++)
                  output+= "Test("+ (i+1) +")\t";
            int x=1;
            foreach (int []course in grades)
            {
                  output+="\n Course["+ x + "]\t";
                  foreach ( int Test in course)
                        output+= Test + "\t";
                  x++;
            }
            // هنا حسبنا أصغر قيمة
            output+=" \n\n Lowest Grade:" + Minimum(grades);
            // و هنا أكبر قيمة
            output+="\n Highest Grade:"+ Maximam(grades);
```

```
            // و هنا المتوسط الحسابي لكل مادة
             x=1;
            foreach (int [] course in grades)
            {
                  output += "\n Average for course ("+ x + ")is :"
                        + Average(course);
                  x++;
            }
            MessageBox.Show(output,"The Grades");
      }

      // وسيلة لحساب أصغر قيمة
     static int Minimum( int[][]grades)
      {
            int lowgrade = 100;
            foreach(int []course in grades)
                  foreach(int test in course)
                        if (test < lowgrade)
                              lowgrade = test;
            return lowgrade;
      }
      //  وسيلة لحساب أكبر قيمة
      static int Maximam( int [][] grades)
      {
            int highgrade=0;
            foreach (int[] course in grades)
                  foreach ( int test in course)
                        if (test > highgrade)
                              highgrade = test;
            return highgrade;
      }
       // وسيلة لحساب المتوسط الحسابي
      static double Average(int [] course)
      {
            int total= 0;
            for (int i=0; i< course.Length ; i++)
                  total += course[i];
            return (double) total/ course.Length;
      }
}
```

و النتيجة كالتالي :



المتحكم params للمعاملات :

هذا المتحكم خاص للمصفوفات و هو يجعل الوسيلة تستقبل معامل من نوع مصفوفة ذات بعد واحد ، و الشيء المميز في هذا المعامل (معامل المصفوفة) هو أنه يستطيع أن يستقبل مجموعة متغيرات و يحولها إلى مصفوفة من نفس النوع ، و بالمثال يتضح المقال :

```
using System;
class Test
{
      static void F(params int []args)
      {
            Console.Write("\nArray contains {0} element:"
,args.Length);
            foreach (int i in args)
                  Console.Write(i+" ");

      }
      static void Main ()
      {
            int [] arr ={ 1,2,3};
            F(arr);
            F(10,20,30,40);// لاحظ ماذا سيحدث هنا
            F();
            Console.ReadLine();
      }
}
```

و النتيجة :

```
Array contains 3 element: 1 2 3
Array contains 4 element: 10 20 30 40
Array contains 0 element:
```

لاحظ كيف استطعنا أن نمرر مجموعة متغيرات لتصبح مصفوفة فيما بعد ، جرب البرنامج السابق بدون المتحكم params لتلاحظ الفرق .و لكن انتبه يجب أن تجعل معامل المصفوفات في آخر قائمة المعاملات.

الواجب :

1- الفرز الفقاعي :

تعد خوارزمية الفرز الفقاعي من أبسط خوارزميات الفرز ، و تقوم على التالي :

1- نأخذ العنصر الأول من المصفوفة و نقارنه بالعنصر الثاني .

2- إذا كان العنصر الأول أكبر من العنصر الثاني نقوم بجعل العنصر الثاني مكان العنصر الأول في حالة الفرز التصاعدي و العكس الفرز التنازلي .

3- نقوم بتكرار الخطوة الثانية مع العنصر الثاني و الثالث و هكذا حتى نصل إلى نهاية المصفوفة لمقارنة العنصر قبل الأخير مع الأخير .

4- نقوم بتكرار الخطوات السابقة (من الخطوة الأولى) ، بحيث يكون عدد مرات التكرار يساوي عدد العناصر في المصفوفة.

و الآن قم بكتابة برنامج ينفذ الخطوات السابقة ، بحيث يقوم بفرز تنازلي و تصاعدي على المصفوفة التالية:

```
a ={ 90,40,88,77,1,5}
```

[إرشاد : استخدم حلقتان تكراريتان لتنفيذ الخطوات السابقة]

2- تحليل البيانات :

تحليل البيانات من أهم تطبيقات الحاسوبية ، و المسألة تقول في إحدى استطلاعات الرأي على أمر ما أدلى 99 شخص بآرائهم على شكل اختيار من 1 إلى 9 و النتيجة كالتالي :

```
6,7,8,9,8,7,8,9,8,9,1,3,5,8,7,7,8,9,4,6,9,9,8,7,8,8,7,5,
4,3,8,2,2,1,1,3,4,8,8,9,5,5,4,3,2,1,7,7,3,1,8,7,6,4,8,8,
9,8,7,5,8,8,7,8,8,7,9,9,8,8,6,5,1,8,8,9,8,7,8,8,6,3,2,8,
8,8,9,7,8
```

و المطلوب التالي :

1- المتوسط الحسابي **Average** (مجموع الأصوات قسمة عددها).

2- القيمة الوسط **Median** (القيمة التي توجد في منتصف الأصوات)

3- رسم بياني يبين تكرر كل عدد من 1 إلى 9 .

4- القيمة الأكثر تكرارا مع بيان كم مرة كررت **Mode** .

ملاحظات :

أ- القيمة الوسط يشترط أولاً فرز الأصوات تصاعديا ثم أخذ القيمة الوسط أي 2 / 99 = 49 ، أي قيمة 49 .

ب – لإيجاد التكرار كل عدد استخدم مصفوفة تحمل الإجابات **answer** (أي من 1 إلى 9) ، ثم ضعها مكان فهرس مصفوفة الأصوات 99 (**frq**)كالتالي :

```
++ frq[answer[j]];
```

ثم قم بحلقة تكرارية تمر على جميع الأصوات (j) ، لينتج مصفوفة فهرسها الإجابات و قيمها عدد مرات تكرر تلك الإجابات.

3 – البحث الخطي و الثنائي :

أ- البحث الخطي : يقوم البحث الخطي على مقارنة الرقم المراد بحث مع كل عناصر المصفوفة حتى يجد ذلك الرقم .

قم بكتابة وسيلة تقوم بالبحث الخطي تستقبل المصفوفة و الرقم المراد البحث عنه و ترجع موقع ذلك الرقم في المصفوفة إن وجد بواسطة الفهرس و المصفوفة هي :

```
a={2,4,6,8,10,12,,14,16,18,22,20,26}
```

ب - البحث الثنائي : قد تكون لاحظت أن البحث الخطي ليس عملي في المصفوفات الضخمة حيث يستهلك وقتا كثيرا حتى يمر على جميع عناصر المصفوفة ، تقوم فكرة البحث الثنائي على فكرة بسيطة تقول إذا كانت عندك مجموعة من الأرقام المرتبة تصاعديا من الأصغر إلى الأكبر ، على سبيل المثال:

1، 2، 6، 8، 10 ، 11 و أنت تريد أن تبحث عن الرقم 8 ، اتبع الخطوات التالية :

1- قم بأخذ الرقم الذي يوجد في منتصف تلك السلسلة و هو 6 .

2- قارن ذلك الرقم مع 8 ، فإذا كان أصغر منه فخذ الرقم الذي يقع في المنتصف بين الرقم الذي يلي 6 و رقم الأخير وقارنه مع 8 ، فإذا كان أكبر منه فاجعل الرقم الذي قبله أي 8 هو أكبر رقم و قم بأخذ الرقم الذي في المنتصف و هو في هذه الحالة 8 .

لربما أصابك إرباك ، و لكن فكرة البحث الثنائي بسيطة جدا ، حيث تقوم بحذف نصف عدد الأرقام التي بحثت فيها ، فلذا يقل مجال البحث كالتالي حيث النجمة تدل على منتصف :

11 10 8 6* 4 2 1	a[(0+6)/2]=a[3]
11 10* 8	a[(4+6)/2]=a[5]
8*	a[(4+4)/2]=a[4]

و الآن قم بكتابة وسيلة تقوم بالبحث الثنائي حيث تستقبل مصفوفة و الرقم المراد البحث عنه و ترجع فهرسه إن وجد ، اختبر هذه الوسيلة مع المصفوفة التالية :

```
a={0,6,5,4,3,2,1,12,13,14,20}
```

****
**
*