الباب الرابع

بناء منطق البرنامج

محتويات الباب

- المنشأ الشرطى if-else
- العبارات الشرطية المتاخلة (Nested if-else)
- الاختيار متعدد البدائل (السويتش) switch
- التعبير الشرطى (Conditional Expression)
- الحلقات التكرارية (Loops)
- الحلقة التكرارية for
- الحلقة while
- الحلقة do-while
- التحكم فى مسار البرنامج
- المصفوفات (Arrays)
- الحلقة التكرارية foreach

استخدام الشروط

نـــناقش فـــى هذه الفقرة كيفية تكوين الشروط فى البرنامج باستخدام التعبيرات الشرطية والعبارات الشرطية.

المؤثرات العلاقية (Relational Operators)

تستخدم المؤثرات العلاقية فى بناء التعبيرات العلاقية المستخدمة فى المقارنة مثل:

```
x > y  // أكبرمن
y < 255  // أصغر من
myInt == 44 // يساوى
myVar >= yourInt // أكبر من أو يساوى
```

والجدول التالى يوضح المؤثرات العلاقية فى لغة سى#.

جدول (1-4) المؤثرات العلاقية

المؤثر	المعنى
>	أكبر من
=<	أكبر من أو يساوى
<	اصغر من
=>	أصغر من أو يساوى
==	يساوى
=!	لا يساوى

وتقيم التعبيرات العلاقية بأحد قيمتين: صحيح (true) أو غير صـــحـــيح (false) ، أى أنها من النمط البوليانى (bool). وبخلاف لغة سى++ فإن القيمة false لا تكافئ 0 وكذلك القيمة true لا يعادلها قيمة غير صفرية.

مثال (4-1)

```
// Example 4 -1.cs
// Relational Operators

using System;
class OperatorClass
{
  static void Main()
  {
  char myChar = 'A';
  int myInt = 55;
  int yourInt = 44;

    Console.WriteLine(myChar ==  'A');
    Console.WriteLine(myInt >= yourInt);
    Console.WriteLine(100 > 100.2);
  }
}
```

تنفيذ البرنامج:

```
True
True
False
```

المؤثرات المنطقية (Logical Operators)

تستخدم المؤثرات المنطقية فى الجمع بين مجموعة من التعبيرات العلاقية (الشروط). ويقيم التعبير المنطقى بأحد قيمتين: صحيح (true) أو غير صحيح (false). والجدول التالى يحتوى على أمثلة لهذه المؤثرات بفرض أن x تساوى 5 و y تساوى 10.

جدول (4-2) المؤثرات المنطقية

المؤثر	الاسم	مثال لتعبير شرطى	النتيجة
&&	الضرب المنطقى المقصور	(x>5) && (y==10)	false (لا يختبر الشرط الثانى)

			بعد فشل الشرط الأول)
||	الجمع المنطقى المقصور	(x==5) || (y==12)	true (لا يختبر الشرط الثانى بعد نجاح الشرط الأول)
&	AND الضرب المنطقى	(x>5) && (y==10)	false (تكون النتيجة true إذا تحقق كل من الشرطين)
|	OR الجمع المنطقى	(x==5) || (y==12)	true تكون النتيجة true إذا تحقق أحد الشرطين)
!	NOT النفى المنطقى	!(x==7)	true (تكون النتيجة true إذا لم يتحقق الشرط)

ملاحظات على الجدول:

- مؤثـرات الضرب المنطقى (& و &&): لاحظ فى المثال الأول أن المؤثر && لا يحتاج إلى إتمام اختبار الشرط الثانى طالما أن الشرط الأول قد فشل ولذلك تسمى هذه العملية بالقصر (Short circuit). أما مع المؤثر & فإنه يختبر طرفى التعبير قبل الخروج بالنتيجة.
- مؤثـرات الجمـع المنطقى (| و ||): بالمثل فإن العملية المثال الثانى تسـمى بالقصر لأنها تنتهى بعد تقييم التعبير الأيسر لأن تحقق شرط واحد يكفى لنجاح العملية.
- المؤثـرات علـى مستوى البت (Bitwise operators): إذا استخدمت المؤثرات & و | مع التعبيرات العددية فإنها تعمل على مستوى البت. كما أن هذه المؤثرات تستخدم مع المؤشرات فى الكود غير المأمون (Unsafe code). ولن نغطى هذا الموضوع فى هذا الكتاب.

المنشأ الشرطى if-else

يستخدم المنشأ الشرطى if-else لبناء العبارات الشرطية ويأخذ الصورة الآتية:

```
if (Condition)
  Statement_1(s);
[else
  Statement_2(s)]
```

حيث:

- Condition: تعبير علاقى (أو منطقى)
- Statement_1(s): عبارة أو "بلوك عبارات" النتيجة الأصلية
- Statement_2(s): عبارة أو "بلوك عبارات" النتيجة البديلة

(لاحظ أن كل مايقع بين العلامتين [] يعتبر اختيارياً)

وكما نرى فى تصميم المنشأ الشرطى أن الكلمة المفتاحية if قد تستخدم بمفردها لبناء المنشأ ، ويتبعها عبارة أو أكثر للتعبير عن النتيجة الأصلية لتحقق الشرط ، مثل:

```
if (salary > 2000)
  Console.Write( "Salary is greater than 2k" );
```

ويجوز أن يضاف إليها الجزء else الذى يحتوى على عبارة أو أكثر للتعبير عن النتيجة البديلة ، مثل:

```
if (salary > 2000)
  Console.Write("Salary is greater than 2k");  // النتيجة الأصلية
else
  Console.Write("Salary is less than 2k");  // النتيجة البديلة
```

وفـــى حالـــة اســـتخدام أكثر من عبارة للتعبير عن إحدى النتيجتين فإن العـــبارات توضـــع بين قوسى بلوك {}. وعلى سبيل المثال فإن تحقق الشرط التالى يؤدى إلى تنفيذ العبارات المتتالية التى تقع بداخل البلوك:

```
if (salary > 2000)
{
  Console.WriteLine("Original result_1");
  Console.WriteLine("Original result_2");
  Console.WriteLine("Original result_3");
}
```

ومن الجائز أيضاً أن تكون عبارة النتيجة الأصلية أو البديلة عبارة عن عبارة شرطية أخرى ، وفى هذه الحالة نحصل على شرط مركب. وفى المثال التالى نختبر حالة اللبنة المدخلة من لوحة الأزرار لنعرف إذا ما كانت حرفاً أم رقماً كالآتى:

- إذا كانت حرفاً فإننا نختبر حالة الحرف لنعرف إذا ماكان:
  - ❖ حرفاً كبيراً (Capital) ، ونطبع الرسالة المناسبة
  - ❖ أوحرفاً صغيراً (Small) ، ونطبع الرسالة المناسبة
- أما إذا لم تكن حرفاً فإننا نطبع الرسالة المناسبة.

مثال (4-2)

```
// Example 4-2.cs
// Character Tester

using System;

public class CharacterTester
{
  public static void Main()
  {
    Console.Write("Please enter a character: ");
```

```
    char yourC  har = (char) Console.Read();
    if (Char.IsLetter(yourChar))
      if (Char.IsLower(yourChar))
        Console.WriteLine("The letter {0} is small.", yourChar);
      else
        Console.WriteLine("The letter {0} is capital.", yourChar);
    else
      Console.WriteLine("The character {0} is not alphabetic.",
                        yourChar);
  }
}
```

تنفيذ البرنامج:

```
Please enter a character: a    // التشغيل الأول
The letter a is small.

Please enter a character: W    // التشغيل الثانى
The letter W is capital.

Please enter a character: 33    // التشغيل الثالث
The character 3 is not alphabetic.
```

ملاحظة على البرنامج:

لاحظ عند قراءة اللبنات بالأسلوب **Read()** أنه يتجاهل ما يلى أول لبنة فالعدد 33 فى التشغيل الثالث قد اعتبر أن اللبنة المدخلة هى 3.

أساليب التعامل مع اللبنات

للتعامل مع اللبنات فى المثال السابق استخدمنا بعض الأساليب من المنشأ **System.Char** بمكتبة دوت.نت وهى:

- لقراءة لبنة من لوحة الأزرار استخدمنا الأسلوب Read() مع استخدام الإسقاط للتحويل إلى نمط اللبنة (char).

- لاختبار الحروف الصغيرة استخدمنا الأسلوب **Char.IsLower()** (يمكنك أيضاً اختبار الحروف الكبيرة بالأسلوب **Char.IsUpper()**)
- لاختبار لبنات الحروف استخدمنا الأسلوب **Char.IsLetter()** (يمكنك أيضاً استخدام الأسلوب **IsDigit()** لاختبار لبنات الأرقام)

للمزيد من المعلومات عن المنشأ Char يمكنك الرجوع إلى الموقع: http://msdn.microsoft.com/.

(للبحث عن فصيلة ما (أو أى عنصر آخر) اكتب اسمها فى صندوق البحث " Search " فيتولى البرنامج عملية البحث عنها وعرضها على الشاشة.)

العبارات الشرطية المتداخلة (Nested if-else)

من الجائز أيضاً أن تتداخل العبارات لتكوين شجرة من الشروط بالصورة الآتية:

```
if (Condition_1)
  Statement_1;
else if (Condition_2)
  Statement_2;
else if (Condition_3)
  Statement_3;
...
else
  Statement_n;
```

وسوف نترك لك بناء برنامج مماثل لمنطق البرنامج السابق باستخدام العبارات الشرطية المتداخلة.

تدريب (4-1)

أعد كتابة البرنامج 4-2.cs مستخدماً العبارات الشرطية المتداخلة.

الاختيار متعدد البدائل (السويتش) switch

يســـتخدم هذا المنشأ فى الحالات التى تحتوى على مجموعة من البدائل مثل اختيار أحد الأدوار أثناء ركوب المصعد ، أو اختيار مشروب معين مـــن ماكينة المشروبات ، أو اختيار عنصر من قائمة برنامج. ونلاحظ فى جميع هذه الحالات انك تضغط على زر واحد لتحقيق غايتك. ويأخذ السويتش الصورة الآتية:

```
switch (variable)
{
  case constant-1:
    statement(s)
    jump-statement
  case constant-2:
    statement(s)
    jump-statement
  case constant-3:
...
...
  [ default:
    statement(s)
    jump-statement]
}
```

حيث:

- variable: متغـــير صـــحيح أو حـــرفى تقوم على أساسه عملية الاختيار
- statement(s): عبارة أو "بلوك عبارات" تنفذ مع الاختيار المعين

- ❖ jump-statement: عبارة تفريع (Branching) لنقل دفة التحكم خارج المنشأ (وتستخدم عادة العبارة **break** أو **goto** كما سيلى)
- ❖ **default**: للتعامل مع جميع الحالات المتبقية ، وهو جزء اختيارى

وتتابع الحالات (**cases**) التى يحتوى عليها المنشأ لتغطى أغلب الاحتمالات الممكنة ، إما ما يتبقى من احتمالات فإن الجزء default يتولاها بما يناسب.

والمثال التالى يحاكى قائمة لاختيار السندوتشات (لحم ، دجاج ، سمك) ، وعندما تدخل رقم الاختيار يخبرك البرنامج عن الثمن المطلوب. ولو أنك أدخلت رقماً غير موجود بالقائمة فإنك تتلقى رسالة تنبهك إلى الأرقام المتاحة وهى 1, 2, 3. وفى كل حالة يطبع البرنامج رسالة شكر للتعامل مع المحل.

مثال (3-4)

```
// Example 3-4.cs
// switch example

using System;

class FastFoodClass
{
  public static void Main()
  {
    Console.WriteLine("Sandwiche s: 1=Beef 2=Fish 3=Chicken");
    Console.Write("Please enter your selection: ");
    string s = Console.ReadLine();
    double totalCost = 0;
    switch(s)
    {
```

```
        case "1":
          totalCost = 4.5;
          break;
        c  ase "2":
          totalCost = 5.25;
          break;
        case "3":
          totalCost = 4.2;
          break;
        default:
          Console.WriteLine("Invalid selection. Please select 1, 2, or
3.");
          break;
      }
      if (totalCost != 0)
        Console.WriteLine("Please pay {0} LE.", totalCost);
      Console.WriteLine("Thank you for your business.");
    }
}
```

أمثلة لتنفيذ البرنامج:

الآتى بعد أمثلة من تشغيل البرنامج:

```
Sandwiches: 1=Beef 2=Fish 3=Chicken    // التشغيل الأول
Please enter your selection: 3    // الرقم المدخل
Please pay 4.2 LE.
Thank you for your business.

Sandwiches: 1=Beef 2=Fish 3=Chicken    // التشغيل الثانى
Please enter your selection: 0    // الرقم المدخل
Invalid selection. Please select 1, 2, or 3.
Thank you for your business.
```

ملاحظة إلى مبرمجى سى++:

هـناك اخـتلاف بين السويتش فى لغة سى# والسويتش فى لغة سى++. ففى لغة سـى++ إذا لم تتضمن الحالة عبارة تفريع فإن الاختيار يتحول تلقائياً إلى الحالة

التالية. أما فى لغة سى# فإن المترجم يعترض على غياب عبارة التفريع. ومع ذلك يمكنك أن تمنح السويتش هذه الخاصية بترك الحالة فارغة من الكود تماماً ، مثل:

```
switch(n)
{
  case 1:  //   تحول تلقائى إلى الحالة رقم ٣
  case 2:  //   تحول تلقائى إلى الحالة رقم ٣
  case 3:
    Con   sole.WriteLine("cases 1 & 2 come here!");
    break;
 // ...
}
```

التعبير الشرطى (Conditional Expression)

يستخدم التعبير الشرطى لبناء منشأ شرطى سريع ، وهو يأخذ الصورة الآتية:

```
Condition ? expression-1 : expression-2
```

حيث:

- Condition: الشرط المطلوب تحققه
- expression-1: تقييم التعبير فى حالة تحقق الشرط
- expression-2: تقييم التعبير فى حالة عدم تحقق الشرط

وبالطبع يمكن تخصيص التعبير الناتج إلى متغير.

انظر المثال التالى للتعبير الشرطى:

```
result = x != 0.0 ? Math.Tan(x) : 1.0;
```

إن هـذه العـبارة تقول "إذا كانت x لا تساوى صفراً ، خصص التعبير Math.Tan(x) للمتغير result ، وإلاّ فخصص القيمة 1.0 للتعبير result". أى أن هـذه العـبارة تكافئ تماماً العبارات التالية التى تستخدام المنشأ الشرطى if-else:

```
if (x != 0.0)
  result = Math.Tan(x);
else
  result = 1.0;
```

وفـى الـبرنامج التالى نضع هذه العبارة فى مثال قابل للتنفيذ. ونلاحظ استخدام بعض أساليب فصيلة الرياضة (**Math**) بمكتبة دوت.نت ، مثل النسـبة المثلثـية **Tan** وأسلوب التقريب **Round** وحقل النسبة التقريبية بالتقدير الدائرى **PI**. ويمكنك الاطلاع على جميع أعضاء الفصيلة Math فى موقع الوب:

http://msdn.microsoft.com

مثال (4-4)

```
// Example 4-4.cs
// Conditional expression example

using System;

class Conditional
{
  public static void Main()
  {
    double x = Math.PI/3, result = 0;
    result = x != 0.0 ? Math.Tan(x) : 1.0;
    Console.WriteLine("Exact value = {0}", result );
    Console.WriteLine("Approximate value = {0}",
                      Math.Round(result ));
  }
}
```

تنفيذ البرنامج:

```
Exact value = 1.73205080756888  // النتيجة الدقيقة
Approximate value = 2    // النتيجة المقربة
```

لاحظ فى المثال السابق أننا قد استخدمنا طريقة مختصرة لإعلان متغيرين وشحنهما فى نفس العبارة:

```
double x = Math.PI/3, result = 0;
```

إن هذه العبارة تكافئ العبارات الآتية:

```
double x = Math.PI/3;
double result = 0;
```

الحلقات التكرارية (Loops)

تستخدم الحلقات التكرارية فى تكرار عملية معينة ، وقد يستمر التكرار عدداً محدداً من المرات أو حتى يتحقق شرط معين أو يستمر بلا نهاية. وهناك أكثر من عبارة لتحقيق الحلقة التكرارية تتميز كل منها بخصائص متفردة.

الحلقة for

تؤدى الحلقة **for** إلى تنفيذ بلوك من العبارات طالما يستمر تحقق شرط معين ، وصورتها كالآتى:

```
for ([initializers]; [expression]; [iterators]) statement(s)
```

حيث:

- initializers: عبارات شحن متغير العداد

- ❖ expression: التعبير الشرطى اللازم لاستمرار الحلقة
- ❖ iterators: عبارات زيادة أو نقصان العداد
- ❖ statement(s): العبارة أو بلوك العبارات التى تتكرر أثناء تحقق الشرط

والمثال التالى يوضح صورة مبسطة لحلقة تكرارية تطبع الأعداد من 1 إلى 5 كل على سطر مستقل:

```
for (int counter = 1; counter <= 5; c  ounter++)
{
  Console.WriteLine(counter);
}
```

فـــى هـــذا المـــثال قد وضعنا العبارة المراد تكرارها بين قوسى بلوك للتوضـــيح ، ولكـــن هـــذا ليس بإلزام. ونلاحظ أيضاً فى هذا المثال أن متغـــير العـــداد "counter" قد تم شحنه بداخل الحلقة التكرارية ، وهذه الخاصـــية موجودة أيضاً بلغة سى++. والبرنامج التالى يضع المثال فى صورة قابلة للتنفيذ.

مثال (5-4)

```
// Example 4-5.cs
// for loop example

using System;

class ForLoop
{
  static void Main()
  {
    for (int counter = 1; counter <= 5; counter++)
    {
      Console.WriteLine  (counter);
```

```
    }
  }
}
```

تنفيذ البرنامج:

```
1
2
3
4
5
```

وفـــى إمكانك أن تستخدم العبارة **continue** لكى تتخطى رقماً معيناً من أرقام الحلقة والاستمرار إلى الرقم التالى ، مثل:

```
if (counter == 2) continue;
```

فى هذه الحالة سوف تقفز الحلقة من الرقم 1 إلى 3 مباشرة. كما يجوز إجهـــاض الحلقـــة تماماً باستخدام العبارة **break** عند تحقق شرط معين مثل:

```
if (counter == 4) break;
```

إن هذا يؤدى إلى إنهاء الحلقة عندما يصل العداد إلى الرقم 4. ونلاحظ فـــى صـــيغة العبارة for أن جميع عناصر العداد اختيارية ، بمعنى أنك تســـتطيع كتابة حلقة تكرارية كالمثال الآتى وهى تستمر بلا نهاية حتى تضغط على مجموعة الأزرار Ctrl+C لإيقاف البرنامج:

```
for (;;)
{
   Console.WriteLine("Hello again!");

}
```

وتتأتى فائدة هذه الحلقة فى التعامل مع المصفوفات كما سيلى.

الحلقة while

تستمر الحلقة **while** فى تنفيذ بلوك من العبارات طالما يستمر تحقق شرط معين. وتأخذ الحلقة الصيغة الآتية:

```
while (condition) statement(s)
```

حيث:

- ❖ condition: الشرط اللازم لاستمرار الحلقة
- ❖ statement(s): العبارة أو بلوك العبارات المطلوب تكرارها

ولأن الحلقة while تبدأ باختبار الشرط فإنها قد لا تنفذ على الإطلاق إذا لم يتحقق الشرط. والمثال الآتى يؤدى إلى طباعة الأعداد من 1 إلى 5 كل على سطر مستقل وهو يستلزم شحن العداد counter بالقيمة الابتدائية 1 قبل بدء الحلقة التكرارية:

```
while (counter  < 6)
{
  Console.WriteLine(counter);
  counter++;
}
```

وفيما يلى نقدم مثالاً آخر يطبع أرقام العداد ولكن تحت شروط جديدة هى:

- العبارة **continue** تؤدى إلى الاستمرار فى الحلقة مع تخطى الرقم 2 الوارد بالشرط:

```
if (counter == 2) continue;
```

- العـــبارة **break** تؤدى إلى الخروج تماماً من الحلقة التكرارية عندما يزيد العداد عن الرقم 5 وفقاً للشرط:

```
if (counter > 5) break;
```

مثال (4-6)

```
// Example 4-6.cs
// while loop example

using System;
class WhileClass
{
  public static void Main()
  {
    int counter = 1;
    while (counter != 0)
    {
      counter++;
      if (co  unter == 2) continue;  //  استمر فى العد
      if (counter > 5) break;  //    اخرج من الحلقة
       Console.WriteLine(counter);
    }
  }
}
```

تنفيذ البرنامج:

```
3
4
5
```

الحلقة do-while

تأخذ هذه الحلقة الصورة التالية:

```
do statement(s) while (condition);
```

حيث:

- condition: الشرط اللازم لاستمرار الحلقة
- statement(s): العبارة أو بلوك العبارات المطلوب تكرارها

وتتميز الحلقة do-while بأنها تنفذ مرة واحدة على الأقل بصرف النظر عن تحقق الشرط من عدمه وذلك لأن الشرط يتم اختباره فى النهاية.

والمـــثال التالى يوضح كيفية استخدام الحلقة do-while ، وكما نرى أن العبارات اللازم تنفيذها تسبق شرط استمرار الحلقة. كما نرى استخدام العـــبارة continue فى تخطى رقم العداد الحالى والقفز إلى العدد التالى. أمـــا العبارة **return** فقد استخدمت هنا لتحقيق نفس الغرض الذى تحققه العبارة break.

مثال (4-7)

```
// Example 4-7.cs
// do-while example

using System;
public class DoWhileClass
{
  static void Main ()
  {
    int x = 0;
    do
    {
      x++;
      if (x%2 != 0)   //    إذا كان العدد فردياً
        continue;      //  استمر
      if (x == 8)     //     إذا وصل العداد إلى8
```

```
        return;      //    (ارجع) اخرج من الحلقة
      Console.WriteLine(x);
    }
      while(x < 100);
  }
}
```

تنفيذ البرنامج:

```
2
4
6
```

ملاحظات على البرنامج:

لاحـــظ أننا استخدمنا هنا **مؤثر باقى القسمة %** لاختبار الأعداد الفردية واستثناءها من الحلقة التكرارية.

```
if (x%2 != 0)
  continue;
```

إن هـــذه العبارة تقول "إذا كان العدد لا يقبل القسمة على 2 فاستمر إلى دورة العـــداد التالـــية. وبالطبع فإذا كان العدد يقبل القسمة على 2 فإن الشرط التالى يصبح true:

```
x%2 == 0
```

كمـــا استخدمنا هنا العبارة return للخروج من الحلقة التكرارية. ولكن هـــذه العـــبارة لا يقتصـــر دورها على كسر الحلقة ، ولكنها تؤدى إلى رجـــوع الأســـلوب (أو الدالـــة) الذى توجد بداخله. معنى ذلك أنك لو وضعت أى عبارة تالية للحلقة فى البرنامج السابق لن تنفذ لأن الأسلوب Main قد انتهى تنفيذه.

ولتجرب الأسلوب الآتى فى البرنامج السابق:

```
static void Main ()
{
```

```
int x = 0;
do
{
  x++;
   if (x == 4)
     return;  // Main    رجوع الأسلوب
  Console.WriteLine(x);
}
while(x < 100);
Console.WriteLine("The loop is done!");     // لن تنفذ هذه العبارة
}
```

إن العبارة الأخيرة التى تطبع الرسالة "The loop is done !" لن تنفذ. ولو إنك استبدلت العبارة return بالعبارة break فسوف تنفذ العبارة الأخيرة وتظهـــر الرسالة على الشاشة. وكما رأينا فى شرح الأسلوب Main أن العبارة return قد تستخدم لإرجاع قيمة عددية ما عند رجوع الأسلوب.

التحكم فى مسار البرنامج

رأيـــنا من قبل أن العبارات التالية يمكن استخدامها للخروج من الحلقات التكرارية:

break
goto
return
throw

وقـــد رأيـــنا أمـــثلة للعبارات return ، break فى الأمثلة السابقة ، أما العبارة goto فإنه لا يوصى باستخدامها لأنها تفسد معمار البرنامج. ومع ذلـــك فقـــد تســـتخدم أحياناً للخروج من عدة حلقات متداخلة. كما أنها تستخدم مع منشأ السويتش للتوجه إلى حالة (case) معينة كالمثال الآتى:

```
case 1:
  // do somethin g
  goto case 3;
```

وبصفة عامة فإن العبارة goto تأخذ إحدى الصيغات الآتية:

```
goto label;
goto case expression;
goto default;
```

حيث:

- label: عنوان يستخدم لتمييز الموضع المعين الذى يتم تحويل مسار البرنامج إليه. (لاحظ فى منشأ السويتش أن كل حالة من الحالات تستخدم عنوانا ما).

والعنوان عبارة عن اسم ما تعقبه العلامة ":" كما فى المثال الآتى الذى يحتوى على العنوان Finish:

```
  goto Finish;
  ...
Finish:    // العنوان
  Console.WriteLine("You are done here!");
```

أما العبارة throw فهى تستخدم فى تتبع واصطياد الأخطاء (Exceptions) ، وسوف يلى شرحها فى فقرة مستقلة.

المصفوفات (Arrays)

تمثل المصفوفة مجموعة من العناصر تنتمى إلى نفس النمط. وقد تكون المصفوفة ذات بعد واحد مثل مجموعة أشخاص يقفون فى صف واحد ،

وقد تكون ذات بعدين مثل مجموعة التلاميذ فى الفصل ، حيث ينتظمون فى صفوف متقاطعة (صفوف وأعمدة). ومن الجائز أن تكون المصفوفة متعددة الأبعاد. كما يجوز أن تكون عناصر المصفوفة عبارة عن مصفوفات ، وفى هذه الحالة نطلق عليها اسم مصفوفة مصفوفات (Jagged array).

المصفوفات ذات البعد الواحد (One-Dimensional Arrays)

تعلن المصفوفة ذات البعد الواحد كالمثال الآتى:

```
int[] myIntArray = new int [10];
```

إن هذا الإعلان يحقق النتائج الآتية:

- حجز مكان لمصفوفة من الأعداد الصحيحة int بالاسم myIntArray.
- تتكون المصفوفة من عشرة عناصر تبدأ من الدليل [0] وحتى الدليل [9]. أى من العنصر myIntArray[0] حتى العنصر myIntArray[9].
- يستخدم المؤثر new فى حجز أماكن المصفوفة و شحن عناصرها إلى القيمة سابقة التعريف ، وهى صفر فى هذه الحالة.

ويمكنك الإعلان عن مصفوفة من أى نمط (قيمة أو مرجع) فالمثال التالى يعلن عن مصفوفة من الحرفيات:

```
string[] myStringArray = new string [10];
```

والفـــرق بيـــن العناصر من نمط القيمة والعناصر من نمط المرجع أن الأولى يتم شحنها بالقيم سابقة التعريف (أنظر الجدول بالملحق ب). أما العناصر من نمط المرجع فإنه يتم شحنها بالقيمة الابتدائية **null**. والقيمة null هـــى عبارة عن مرجع لا يشير إلى أى شىء ، ولذلك سوف نطلق عليه اسم المرجع الفارغ.

شحن المصفوفة عند الإعلان عنها

مـــن الجائز أن تشحن المصفوفة بالقيم فى نفس عبارة الإعلان كالمثال التالى:

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

وفى هذه الحالة لا نحتاج إلى إعلان عدد عناصر المصفوفة.

والمثال التالى لمصفوفة حرفيات تمثل أيام الأسبوع:

```
string[] myDay = new string []
                {"Sun","Sat","Mon","Tue","Wed","Thu","Fri"};
```

وهناك صور مختصرة للإعلانات ، حيث نستغنى عن الجزء المحتوى على المؤثر new على أساس أن النمط بديهى ويمكن للمترجم استنتاجه. أنظر المثال الآتى:

```
int[] myIntArray = {1, 2, 3, 4, 5};
string[] myDay = {"Sun","Sat","Mon","Tue","Wed","Thu","Fri"};
```

كمـــا أنـــه من الممكن الإعلان عن متغير مصفوفة بدون شحنه ، ولكنه لابد من استخدام المؤثر new عند التخصيص للمتغير ، مثل:

```
int[] myIntArray;  // الإعلان بدون شحن

myIntArray = new int[] {1, 3, 5, 7, 9}; // الشحن والتخصيص للمصفوفة
```

المصفوفات متعددة الأبعاد (Multi-Dimensional Arrays)

تـــؤدى العبارة التالية إلى إعلان وشحن مصفوفة من الأعداد الصحيحة ذات بعدين [3,2] بالاسم myTwoDimArray:

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

كما تؤدى العبارة التالية إلى إعلان وشحن مصفوفة من الحرفيات ذات بعدين [4,4] بالاسم grades:

```
string[,] grades = new string[4, 4] { {"Pass"," Good", "VeryGood",
"Distinct"}, {"55%","65%", "75%", "85%"} };
```

وكمـــا مـــع المصـــفوفات ذات البعد الواحد ، فمن الجائز إغفال أبعاد المصفوفة والاكتفاء بشحنها كالآتى:

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

وفى هذه الصورة قد أغفلنا الأبعاد لمصفوفة الحرفيات أيضاً:

```
string[,] grades = new string[,] { {"Pass"," Good", "VeryGood",
"Distinct"}, {"55%","65%", "75%", "85%"} };
```

كما يجوز أيضاً إغفال عبارة المؤثر new كالآتى:

```
int[,] myTwoDimArray = { {1,2}, {3,4}, {5,6} };
string [,] grades = { {"Pass", "Good", "VeyGood", "Distinct"},
{"55%","65%", "75%", "85%"} };
```

مصفوفات المصفوفات (Jagged Arrays)

لإعلان وشحن مصفوفة من المصفوفات بالاسم myJaggedArray ذات أبعاد [5][2] فإننا نستخدم العبارة الآتية:

```
int[][] myJaggedArray =
      new int[2][] { new int[] {2,3,4}, new int[] {5,6,7,8,9} };
```

كما يجوز إغفال بعد المصفوفة الأول كالآتى:

```
int[][] myJaggedArray =
      new int[][] { new int[] {2,3,4}, new int[] {5,6,7,8,9} };
```

كما يجوز حذف المؤثر new (الأول) كالآتى:

```
int[][] myJaggedArray =
            { new int[] {2,3,4}, new int[] {5,6,7,8,9} };
```

التوصل إلى أعضاء المصفوفات

بالتوصل إلى أعضاء المصفوفة يمكنك معالجة عناصرها علاوة على طباعتها. وعلى سبيل المثال يمكنك تغيير محتويات العنصر رقم [2] فى المصفوفة ذات البعد الواحد myIntArray باستخدام العبارة:

```
myIntArray[2] = 45;
```

كما يمكن تغيير محتويات العنصر رقم [0,0] بالمصفوفة ذات البعدين myTwoDimArray باستخدام العبارة:

```
myTwoDimArray[0,0] = 133;
```

والعبارات التالية تتوصل إلى أعضاء المصفوفة myJaggedArray وتخصص الرقم 11 للعنصر الأول من المصفوفة الأولى وتخصص الرقم 22 للعنصر الثالث من المصفوفة الثانية:

```
myJaggedArray[0][0] = 11;
myJaggedArray[1][2] = 22;
```

والأداة السهلة فى التخصيص أو القراءة هى الحلقات التكرارية. فلطباعة محتويات المصفوفة myIntArray مثلاً نستخدم الحلقة التالية:

```
for (int i = 0; i <= 4; i++)
  Console.WriteLine(myIntArray[i]);
```

وفى حالة المصفوفة ذات البعدين (myTwoDimArray) فإنه يلزم حلقتان متداخلتان تختص كل واحدة ببعد من أبعاد المصفوفة ، أى:

```
for (int i = 0; i <= 2; i++)
  for (int j = 0; j <= 1; j++)
    Console.WriteLine(myTwoDimArray[i,j]);
```

ولا تنس عند شحن العدادات أن دليل المصفوفة يبدأ دائما من 0. وفى المثال التالى نقدم طريقة طباعة مصفوفة المصفوفات (myJaggedArray) التى شحناها فى الفقرة السابقة ، وذلك بعد تغيير محتويات اثنين من عناصرها.

مثال (4-8)

```
// Example 4-8.cs
// Jagged arrays example

using System;

class JaggedClass
{
  static void Main ()
  {
    int[][] myJaggedArray =
          { new int[] {2,3,4}, new int[] {5,6,7,8,9} };
    myJaggedArray[0][0] = 11;  //  تغيير أول عنصر فى المصفوفة الأولى
```

```
    myJaggedArray[1][2] = 22;  // تغيير ثالث عنصر فى المصفوفة الثانية

    // Print first array:
    Console.WriteLine("First ar ray:");
    for (int j = 0; j <= 2; j++)
    {
      // طباعة أول مصفوفة
      Console.WriteLine(myJaggedArray[0][j]);
    }
    // Print second array:
    Console.WriteLine("Second array:");
    for (int j = 0; j <= 4; j++)
    {
      // طباعة ثانى مصفوفة
     Console.WriteLine(myJaggedArray[1][j]);
    }
    Console.WriteLine("The job is done!");
  }
}
```

تنفيذ البرنامج:

```
First array:
11      // عنصر جديد
3
4
Second array:
5
6
22      // عنصر جديد
8
9
The job is done!
```

ولا يفوتـــنا أن نلاحـــظ أننا قد استخدمنا بعض الأقواس {} الزائدة عن الحاجة لتسهيل قراءة البرنامج.

تدريب (4-2)

اطبع محتويات المصفوفة الحرفية ذات البعدين "grades" بحيث تحصل على النتيجة الآتية عند تشغيل البرنامج:

```
Grade=Pass       Score=55%
Grade=Good       Score =65%
Grade=VeryGood   Score=75%
Grade=Distinct   Score=85%
```

استخدام البارامترات عند تشغيل البرنامج (Arguments)

مـــن خصائص الأسلوب Main أنه يمكن تعديل صورته لكى تستطيع إدخال بارامترات معينة أثناء تنفيذ البرنامج ، وذلك كبديل لقراءة لوحة الأزرار. وهذه هى الصورة الجديدة للأسلوب Main:

```
static void Main(string[] args)
```

وتكتـــب بارامـــترات البرنامج فى صورة حرفيات متتابعة (مصفوفة حرفـــيات). وتســـتقبل البارامترات كحرفيات ، ولذلك يلزم تحويلها إلى الصـــورة المناسبة عند استخدامها. ويتم التحويل بإحدى الطرق السابق شرحها فى الفقرة السابقة.

مثال (4-9):

يستخدم هذا المثال بارامترين عند التشغيل وهما عددان الأول طويل (long) والـــثانى حقـــيقى مضـــاعف الدقة (double) ، ويطبع البرنامج العددين علاوة على حاصل ضربهما.

```
// 4-9.cs
// Parsing arguments

using System;

public class ParseClass
{
  static v oid Main(string[] args)
  {
    long myLong = Convert.ToInt64(args[0]);
    double myDouble = Convert.ToDouble(args[1]);
    double result = myLong * myDouble;

     Console.WriteLine("Your long number is: {0}", myLong);
     Console.WriteLine("Your d  ouble number is: {0}", myDouble);
     Console.WriteLine("The result of multiplication is: {0}",
                       result);
  }
}
```

مثال للتنفيذ:

بفرض أن اسم البرنامج هو example استخدم الأمر التالى لتشغيله:

```
>example 2 1.1
```

نتيجة التنفيذ:

```
Your long number is: 2
Your double number is: 1.1
The result of multiplication is: 2.2
```

استخدام أساليب دوت نت مع المصفوفات

إن المصـــفوفات فـــى لغة سى# ما هى إلاّ هدف ينحدر من الفصيلة System.Array ولذلك فإنه من الممكن استخدام الأساليب والخواص التى تمدنـــا بهـــا هذه الفصيلة لمعالجة المصفوفات بشتى الطرق. وفيما يلى

نعـرض بعـض الخواص والأساليب التى قد تحتاج إليها عند معالجة المصفوفات:

طول المصفوفة Length

تطبع العبارات التالية عـلى الشـاشة الرقم 40 وهو طول المصفوفة (Length):

```
int[,] myArray= new int[10,4];  // إعلان المصفوفة
Console.WriteLine(myArray.Length);  // إيجاد وطباعة الطول
```

رتبة المصفوفة Rank

أما العبارة التالية فهى تطـبع الـرقم 2 الذى يمثل رتبة المصفوفة (Rank):

```
Console.WriteLine(myArray.Rank);  // إيجاد وطباعة الرتبة
```

فرز المصفوفة Sort()

اعتبر المصفوفة العددية الآتية:

```
int[] arr = {3,4,56,8};
```

لكى تفرز هذه المصفوفة (ترتب أعدادها) فإنك تستخدم الأسلوب Array.Sort() ، أى:

```
Array.Sort(arr);
```

عكس المصفوفة Reverse()

لكـى تعكـس عناصـر المصـفوفة السابقة فإنك تستخدم الأسلوب

Array.Reverse() ، أى:

```
Array.Reverse(arr);
```

الحلقة التكرارية foreach

تستخدم الحلقة التكرارية foreach للتوصل إلى أعضاء المصفوفات بشتى أنواعها بدون الحاجة إلى عدّادات وبصرف النظر عن أبعاد المصفوفة ، وهى تأخذ الصورة الآتية:

```
foreach (type identifier in expression) statement(s)
```

حيث:

- type: اسم نمط مثل int أو string
- identifier: اسم متغير
- expression: اسم المصفوفة أو المجموعة (Collection)
- statement(s): العبارة أو بلوك العبارات المطلوب تنفيذها

وعلى سبيل المثال اعتبر المصفوفة العددية الآتية:

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

يمكنك طباعة جميع عناصر المصفوفة بالعبارة الآتية:

```
foreach(int i in myIntArray)
  Console.Write("{0} ",  i);
```

والنتيجة هى:

```
1 3 5 2 4 6
```

ونلاحظ أنه لا فرق بين أن تكون المصفوفة ذات بعد واحد أو متعددة الأبعاد لأن العبارة foreach تتوصل إلى جميع العناصر

بالترتيب. فلو كانت المصفوفة ذات بعد واحد مثل:

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

فإنها تطبع بنفس العبارة:

```
foreach(int i in myIntArray)
  Console.Write("{0} ",  i);
```

ولو كانت المصفوفة مصفوفة حرفيات فإن العبارة foreach تصبح:

```
foreach(string s in myIntArray)
  Console.Write("{0} ",  s);
```

مثال (10-4)

فـــى المثال التالى نستخدم مصفوفة من الحرفيات ونطبعها باستخدام العبارة foreach ، كما نطبع عدد العناصر ورتبة المصفوفة.

```
// Example 4-10.cs
// foreach example

using System;

class foreachClass
{
  static void Main ()
  {
  string [,] nameArray =
            { {"Hazem","Sally"    }, {"Isabella","Pille"} };
  foreach(string n in nameArray )
    Console.Write("{0} ",n);
// اترك سطراً خالياً
  Console.WriteLine();
```

```
// عدد العناصر
  Console.WriteLine("Number of array elements = {0}",
                    nameArray.Le      ngth);
// أبعاد المصفوفة (الرتبة)
  Console.WriteLine("Dimensions of the array = {0}",
                    nameArray.Rank);
  }
}
```

تنفيذ البرنامج:

```
Hazem Sally Isabella Pille
Number of array elements = 4
Dimensions of the array = 2
```

مثال (4-11)

فى المثال التالى نستخدم مصفوفة أعداد ونعالجها باستخدام الأساليب Array.Sort() و Array.Revers() ثـــم نطبعها قبل وبعد تطبيق الأساليب عليها.

```
// Example 4-11.cs
// Array methods example

using System;

class MyClass
{
  static void Main()
  {
    int[] a  rr = {1,4,25,3};

  // Print array:
  foreach (int a in arr)
    Console.Write("{0} ", a);
  Console.WriteLine("Original");

  // Sort, then print:
```

```
  Array.Sort(arr);
  foreach (int a in arr)
    Console.Write("{0} ", a);
  Console.WriteLine("So rted");

  // Reverse, then print:
  Array.Reverse(arr);
  foreach (int a in arr)
    Console.Write("{0} ", a);
  Console.WriteLine("Reversed");
  }
}
```

تنفيذ البرنامج:

```
1 4 25 3 Original
1 3 4 25 Sorted
25 4 3 1 Reversed
```