

الفصل الثاني

المصفوفات Arrays

اقتصر حديثنا في الكتاب الأول من السلسلة على متغير يحفظ قيمة وحيدة أيا كان نوعها، ترى ماذا لو أن لدينا عدة قيم ممثلة لنفس العنصر؟ كجدول يشتمل على درجات الطلاب مطلوب ترتيبها أو استخراج علاقة بينها، هذا ما سنحاول الإجابة عليه في هذا الفصل. بالانتهاء من هذا الفصل ستكتسب المعارف وتتدرب على المهارات التي تجعلك قادرا على:

- مفهوم المصفوفات.
- الإعلان عن المصفوفات.
- المصفوفات متغيرة الحجم.
- المصفوفات متعددة الأبعاد.
- استخدام الدورات للتعامل مع المصفوفات.

ما هي المصفوفة؟

المصفوفة Array هي مجموعة من المتغيرات من نفس النمط ترتبط مع بعضها وتسمى بنفس الاسم، ويسمى كل منها عنصر أو Element ولكل متغير داخل المصفوفة دليل Index مختلف يحدد ترتيبه داخل المصفوفة. وعند استدعاء أي متغير فيها يتم الإشارة إليه باستخدام الدليل Index أو الرقم المسلسل الخاص به داخل المصفوفة. لتقريب المعنى تخيل المصفوفة كعمود رأسي له اسم مميز، ولكنه مقسم إلى خانات لكل خانة رقم مختلف يدل عليها.

يسمح Visual Basic للمبرمج بتحديد عدد العناصر في المصفوفة، فيمكنه جعلها خاوية تماما، لا تحتوي على أي عنصر، كما يمكنه وضع العدد الذي يريد، إضافة إلى ذلك يتيح Visual Basic ما لا يتيح معظم لغات البرمجة الأخرى وهو إمكانية تغيير عدد العناصر في المصفوفة أثناء تشغيل البرنامج.

الإعلان عن المصفوفات

يسمى الإعلان عن المصفوفة أيضا "تحديد أبعاد المصفوفة" **Array Dimensioning**، وعبارة الإعلان عن المصفوفة تشبه عبارة الإعلان عن المتغير العادي، وصيغتها كالتالي:

Dim | Public | Private ArrayName(Subscript) As DataType

وعن هذه الصيغة نوضح ما يلي:

• يتم الإعلان عن المصفوفة بإحدى العبارات: **Dim** أو **Public** أو **Private** وفيما يلي نوضح استخدام كل منها:

▪ **Private**: تجعل المصفوفة محلية إذا تم الإعلان عنها في إجراء فإنها لا ترى إلا في هذا الإجراء، وإن تم الإعلان عنها في قسم الإعلان العام في نموذج أو وحدة برمجية فإنها لا ترى إلا في هذا النموذج أو هذه الوحدة.

▪ **Public**: لا تستخدم إلا في قسم الإعلان العام، وتعني أن المتغير يصبح متاحا في أي مكان من البرنامج (أي وحدة برمجية أو إجراء أو نموذج).

▪ **Dim**: تشبه **Private** عند استخدامها في إجراء أو في قسم الإعلان العام في نموذج، بينما تقوم بعمل **Public** عند استخدامها في قسم الإعلان العام.

• **ArrayName**: اسم المصفوفة

• **Subscript**: أقصى قيمة للدليل، وهذا العدد يحدد عدد العناصر في المصفوفة،

ودائماً يبدأ الدليل بالقيمة 0، معنى هذا إذا تم استخدام الصيغة: **Subscript=5**

فإن عدد العناصر يكون 6.

• **Datatype**: نمط عناصر المصفوفة، وهو أي نمط متاح في **Visual Basic** مثل

Integer أو **Double**.

للتوضيح نسوق المثال التالي:

إذا أردت تصميم برنامج لشئون الموظفين بشركتك وبفرض أن عدد الموظفين في الشركة 100 موظف، استخدم مصفوفة تتكون من 100 عنصر، والصيغة التي تحقق هذا الغرض

لهذه المصفوفة كما يلي:

Public EmpNames(99) AS String

وعن هذه الصيغة نوضح ما يلي:

- لأننا نريد أن نتعامل مع أسماء الموظفين في أي مكان من البرنامج (نموذج أو وحدة برمجية أو إجراء)، فقد استخدمنا الأمر **Public** للإعلان عن المصفوفة باعتبارها مصفوفة عامة.
- اسم المصفوفة هو **EmpNames**.
- إجمالي عدد عناصر المصفوفة (عدد الموظفين) هو ١٠٠ وتكتب ٩٩ بين قوسين حيث أن المصفوفة تبدأ دائماً من الصفر.
- متغيرات هذه المصفوفة من النوع **String** حيث أن الاسم عبارة عن سلسلة من الحروف.

ومن السمات الجديدة التي ظهرت في الإصدار السابق من **Visual Basic** إمكانية كتابة عدد عناصر المصفوفة بدءاً من الدليل 0 وحتى آخر دليل في المصفوفة هكذا مثلاً:

Public EmpNames(0 To 99) AS String

فكلا الصيغتين له نفس المدلول، إلا أن الأخير أسهل في القراءة والفهم. وعندما تريد أن تتعامل مع الموظف رقم ١ في المصفوفة وبفرض أن اسمه "وليد عبدالرازق" فإنك ستكتب الأمر التالي:

EmpNames (0)= " وليد عبدالرازق "

وللإشارة إلى الموظف رقم ٢ وهو "محمد وليد" يتم كتابته بنفس الطريقة كما يلي:

EmpNames (1)= " محمد وليد "

المصفوفات متغيرة الحجم Dynamic Arrays

كما لاحظت في المثال السابق أننا استخدمنا مصفوفة عدد عناصرها ثابت وتسمى **Fixed Size Array**، وعند الإعلان عن هذا النوع من المصفوفات يقوم **Visual**

Basic بحجز مساحة من الذاكرة تتسع لجميع عناصر المصفوفة، فإذا كانت عناصر المصفوفة غير مستغلة كلها فان هذا يعنى استخدام سئ للذاكرة. مشكلة أخرى في المصفوفات ذات العناصر الثابتة وهي أن بعض التطبيقات لا تعرف مقدما عدد العناصر التي ستحتاجها. والحل في هذه الحالة هو استخدام المصفوفة الديناميكية أو المصفوفات متغيرة الحجم. تعتمد فكرة المصفوفة ذات الحجم المتغير على استخدام عدد من العناصر يتحدد بحسب حاجة البرنامج فقط.

بدايةً، يجب الإعلان عن هذه المصفوفات بدون تحديد لعدد العناصر، وإلا لن يتم السماح بتغييرها، كما في السطر التالي:

Dim iScores() As Integer

هذه المصفوفة لا تحتوي على أية عناصر، وأي استخدام لها سيسبب ظهور خطأ، ولكن يجب تحديد أبعادها قبل الاستخدام بعبارة **ReDim** وصيغتها كالتالي:

ReDim [Preserve] ArrayName(Subscript)

- تبدأ العبارة بكلمة **ReDim** وهي عبارة برمجية وليست إعلاناً أي يجب استخدامها داخل إجراء ولا يمكن استخدامها في قسم الإعلان العام.
- كلمة **Preserve** كلمة اختيارية وتعني الاحتفاظ بالعناصر الموجودة حالياً في المصفوفة. نظراً لأن عبارة **ReDim** يمكن استخدامها أكثر من مرة مع نفس المصفوفة فقد يتم استخدامها والمصفوفة بها عناصر بالفعل، لو لم يتم وضع **Preserve** فإن العناصر كلها سيتم تخصيصها بالقيمة صفر، أما إذا تم استخدام **Preserve** فيتم الاحتفاظ بالقيم الحالية، لو تم زيادة عدد العناصر، أما إذا تم تقليل العدد فإن العبارة ستحتفظ بأكثر عدد ممكن من العناصر (العناصر المتعدية لأقصى دليل سيتم حذفها).

- بقية عناصر العبارة تشبه عبارة الإعلان التي سبق وأن شرحناها.

مثال:

يتم الإعلان عن مصفوفة متغيرة الحجم بدون تحديد حجمها كما في المثال التالي:

Dim iScores() As Integer

بعد ذلك يتم تحديد عدد العناصر كالتالي:

ReDim iScores(9)

حيث أن العدد المتوقع هو ١٠ (من صفر حتى ٩).

ويتم ملء عناصر المصفوفة كما يلي:

iScores(0)=1030

iScores(1)=1015

...

iScores(9)=1300

لو أردنا أن نضيف عنصر إلى هذه المصفوفة الآن مع الاحتفاظ بالقيم السابقة يمكننا كتابة السطر التالي:

ReDim Preserve iScores(10)

سيتم الإبقاء على القيم السابقة، مضافاً إليها عنصر فارغ جديد هو iScores(10) يمكن استخدامه كالتالي:

iScores(10)=300

لا يستخدم الأمر Redim إلا داخل إجراء فقط، أي لا يستخدم في قسم الإعلانات مثل الأمر Dim.



المصفوفات متعددة الأبعاد Multidimensional Arrays

كل ما ذكرناه حتى الآن يخص المصفوفة ذات البعد الواحد، أي أنها تستخدم صف واحد من المتغيرات. يتيح لك Visual Basic أن تقوم بتعريف مصفوفات يصل حجمها حتى ٣٢ بعداً!! على كل حال المصفوفة ذات البعدين (المشابهة للجدول) كافية تماماً لمعظم الأغراض البرمجية. وعلى أقصى تقدير لا أظن أنك ستحتاج إلى مصفوفة لها أكثر من ثلاثة أبعاد.

المصفوفة ذات البعدين تشبه الجدول تماماً، يمكن الوصول إلى أي عنصر فيها باستخدام دليلين وليس دليل واحد. الأول هو رقم العمود ، الثاني هو رقم الصف.
للإعلان عن مصفوفة ذات بعدين، نستخدم الصيغة التالية:

Dim|Public|Private ArrayName(SubCols,SubRows) As DataType

نلاحظ هنا استخدام رقمين يحددان الحد الأقصى للدليلين أي يحددان أبعاد هذه

المصفوفة. مثلا للإعلان عن مصفوفة مكونة من ٧ صفوف و ٥ أعمدة (مواقيت الصلاة في أسبوع مثلا) نستخدم عبارة الإعلان التالية:

Dim PrayTimes(4, 6) As Date

حيث:

PrayTimes : هو اسم المصفوفة

4 : أقصى دليل لعدد عناصر أعمدة المصفوفة (الجدول)

6 : أقصى دليل لعدد عناصر صفوف المصفوفة (الجدول)

Date : أي أن المصفوفة من النوع **Date** "تاريخ"

ويكون مجموع العناصر داخل المصفوفة هو ٣٥ عنصرا، أي حاصل ضرب البعد الأول (عدد الأعمدة) في البعد الثاني (عدد الصفوف)

أما لو أردنا أن نعلن عن مصفوفة ثلاثية الأبعاد، ولتكن متاهة مجسمة مثلا كل عنصر فيها عبارة عن كتلة (موجودة أم لا)، فإننا نستخدم الصيغة:

Dim Maze3D(20, 20, 20) as Boolean

لو أردنا بعد ذلك تعريف فضاء المتاهة بوضع الكتل المكونة لها، نستخدم عبارات تخصيص مباشرة كما يلي:

Maze3D(0,0,0) = False

Maze3D(0,0,1) = True

Maze3D(0,1,0) = False

...

Maze3D(20,20,20) = False

المصفوفات المتعددة الأبعاد تشبه أحادية البعد من حيث الاستخدام، إذ ينطبق عليها استخدام صيغة (minIndex To maxIndex) عند تعريف أحد الأبعاد.

كذلك ينطبق على هذا النوع استخدام العبارة **ReDim** إذ يمكن تغيير الأبعاد كأى

مصفوفة أحادية البعد. الأمر المهم هنا أنك لو استخدمت **Preserve** مع المصفوفة

متعددة الأبعاد، فإن البعد الأخير فقط هو الذي يمكن تغيير عدد عناصره.

استخدام الدورات للتعامل مع المصفوفات

لعل أقرب العبارات إلى المصفوفات في البرمجة هي الدورات (Loops). ولا يخلو برنامج يستخدم المصفوفات من دورات تدعم استخدامها. ولعل معالجة المصفوفات من أهم تطبيقات الدورات، حيث يتم عادةً استخدام عداد الدائرة كدليل للمصفوفة التي يتم معالجتها. فيما يلي مثال على استخدام الدورات لمعالجة المصفوفات.

إنشاء واجهة البرنامج

١. قم بإنشاء مشروع نوافذ جديد باسم مناسب وليكن Arrays .
٢. أعد تسمية النموذج الرئيسي إلى frmArrays واجعل عنوانه "المصفوفات" من خلال الخاصية Text وقم بتغيير قيمة الخاصية RightToLeft إلى True .
٣. قم بإضافة مربعي نصوص باسم txtList و txtSum أولهما متعدد السطور (MultiLine=True). قم بتغيير قيمة الخاصية TextAlign في كلا المربعين إلى Center .
٤. قم بإضافة ثلاثة أزرار أوامر أسماؤهم بالترتيب (من أعلى لأسفل) btnRnd و btnSer و btnSum واختر لهم العناوين الموضحة بشكل ١-٢ .
٥. قم بإضافة زر أمر رابع لإغلاق النموذج باسم btnExit وعنوان "خروج".
٦. عند اكتمال عناصر النموذج يجب أن يظهر كما في شكل ١-٢ التالي.



شكل ١-٢ عناصر النموذج في المثال بعد إتمامها

كما ترى تتكون نافذة النموذج من زر لتوليد مجموعة من الأرقام العشوائية وآخر لأرقام متسلسلة، وبعد التوليد يتم إظهار الأرقام في مربع النص الطويل العلوي. إذا قمت بالضغط على زر جمع الأرقام فإن المجموع يظهر في المربع السفلي.

كتابة كود البرنامج

لكتابة الكود الخاص بالنموذج، قم بكتابة الكود التالي في نافذة الكود كما يلي: (تجد البرنامج على القرص المرفق بالكتاب باسم Arrays.vbp داخل المجلد Arrays).

١. قم بكتابة الإعلانات الآتية داخل جزء الإعلانات الخاص بالنموذج:

```
Dim i As Integer, Sum As Integer
Dim inNums(9) As Integer
```

٢. انقر زر btnRnd نقرًا مزدوجًا ثم قم بإدخال الكود التالي داخل إجراء حدث نقر الزر:

```
Private Sub btnRnd_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnRand.Click
txtList.Text = ""
For i = 0 To 9
inNums(i) = Int((100 - 0 + 1) * Rnd() + 0)
txtList.Text = txtList.Text & inNums(i) & vbCrLf
Next i
End Sub
```

٣. انقر زر btnSer نقرًا مزدوجًا ثم قم بإدخال الكود التالي داخل إجراء حدث نقر الزر:

```
Private Sub btnSer_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnSer.Click
txtList.Text = ""
For i = 0 To 9
inNums(i) = i
txtList.Text = txtList.Text & inNums(i) & vbCrLf
Next i
End Sub
```

٤. انقر زر btnSum نقرًا مزدوجًا ثم قم بإدخال الكود التالي داخل إجراء حدث نقر الزر:

```
Private Sub btnSum_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnSum.Click
Sum = 0
For i = 0 To 9
Sum = Sum + inNums(i)
Next i
txtSum.Text = Sum
End Sub
```

٥. انقر زر `btnExit` نقرأ مزدوجاً ثم قم بكتابة كلمة `End` التي تتسبب في إغلاق النموذج وإنهاء التطبيق كما يلي:

```
Private Sub btnExit_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles btnExit.Click  
End  
End Sub
```

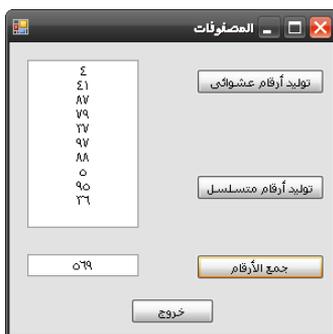
يستخدم البرنامج متغيرين صحيحين (عداد ومجموع) ومصفوفة تحتوي على عشرة عناصر. بالنسبة لتوليد الأرقام هناك التوليد المتسلسل الذي يضع قيمة الدليل في كل عنصر (من ٠ إلى ٩) وهناك التوليد العشوائي الذي يضع رقم يقع بين ٠ و ١٠٠٠ في كل عنصر من العناصر العشرة. كلا النوعين يستخدم دارة يقوم فيها بتوليد الرقم ثم إدراجه في مربع النص المسمى `txtList`. الجدير بالملاحظة هنا طريقة توليد الرقم الصحيح بين ٠ و ١٠٠٠، استخدمنا هنا المعادلة التي يمنحنا إياها ملف المساعدة في الدالة `Rnd` (أي `Random` أو عشوائي) والتي يمكنها توليد أرقام صحيحة بين `LowerN` و `UpperN` لاحظ أن `Rnd` نفسها تولد أرقام كسرية بين صفر و ١).

الصيغة العامة للمعادلة كما يلي:

```
Int((UpperN - LowerN + 1) * Rnd + LowerN)
```

نفذ البرنامج ثم اضغط على زر الأمر الأول لتوليد الأرقام العشوائية، ستظهر مجموعة من الأرقام العشوائية في مربع النص الكبير. عندئذ اضغط على زر الأمر الخاص بجمع الأرقام، سيظهر المجموع في مربع النص السفلي كما في شكل ٢-٢.

قم بتكرار ما سبق ولكن بنقر زر الأمر الثاني، وحينئذ تظهر الأرقام من صفر إلى ٩ بمربع النص العلوي. انقر زر "جمع الأرقام" تحصل على مجموع هذه الأرقام بمربع النص السفلي وهي ٤٥ في هذه الحالة (انظر شكل ٢-٣).



شكل ٢-٢ شكل البرنامج أثناء عمله بعد توليد مجموعة أرقام عشوائية وجمعها.



شكل ٢-٣ شكل البرنامج أثناء عمله بعد توليد مجموعة أرقام متسلسلة وجمعها.

انقر زر "خروج" لإغلاق النموذج وإنهاء التطبيق.