بنى المعطيات

Data Structures

الكلمات المفتاحية

البنية، المصفوفة، الصندقة وفكّ الصندقة.

ملخص الفصل

يهدف الفصل إلى التعرّف على المصفوفة كنمط معطيات مرجعي مشتقّ من الصفّ Array الذي يحتوي على مجموعة من الطرائق التي سهّلت التعامل مع المصفوفات. ويوضّح الفصل مفهوم البنية كأحد أنماط معطيات القيمة، ويعرض البنية الهامة DateTime. ويبيّن كيفية تغليف متغيّر قيمة بغرض من خلال ما يُعرف بالصندقة.

الأهداف التعليمية

يتعرّف الطالب في هذا الفصل على:

- الصفّ Array وبعض طرائقه الشهيرة
- البنية strcut وكيفية استخدامها
- البنية DateTime
- مفهوم الصندقة

مقدمة

نتيجة للأهمية الكبرى التي تتمتّع بها المصفوفات، تمّ توفير عدة طرائق جاهزة للتعامل بسهولة مع عناصر المصفوفة. وبسبب التكلفة الكبيرة لإنشاء أغراض من الصفوف، قد نلجأ لبنية أخفّ وأقلّ تكلفة يمكن اللجوء إليها في بعض الحالات التي يُطلب من الصفّ التعامل بشكل أساسي مع المعطيات، وفي مثل هذه الحالات يكون استخدام البنية `struct` حلّاً مناسباً.

1. المصفوفات

المصفوفة هي بنية معطيات تعبّر عن مجموعة من العناصر التي لها نفس النمط بحيث يمكن الوصول إلى أي من عناصرها باستخدام اسم المصفوفة والرمز الدليلي المقابل للعنصر. وتكون عناصر المصفوفة مفهرسة وفقاً لقيم من النمط int ومتزايدة ابتداءً من القيمة صفر التي تُعتبر الرمز الدليلي لأوّل العناصر.
لقد مرّ معنا في مقرّرات سابقة كيفية التصريح عن مصفوفة والتعامل مع عناصرها، وكانت عناصر المصفوفة ذات نمط بسيط كما في المثال الآتي الذي يوضّح كيفية التصريح عن مصفوفة أعداد من النمط int.

```
int[] arrayInt = { 1 , 2 , 3 };
arrayInt[1] = 5;
```

ومن أجل كلّ صفّ نقوم بإنشائه، يمكن تعريف مصفوفة لتخزين أغراضه، أي يمكن تعريف مصفوفة لأغراض تمّ إنشاؤها من هذا الصفّ.
وكلّ مصفوفة يتمّ تعريفها في لغة C#، ترث من الصفّ Array الذي يُعدُّ الصفّ الأساس لجميع المصفوفات. وبذلك يمكنها استدعاء الطرائق المعرّفة ضمنه، وهذا ما يجعل التعامل مع المصفوفات أمراً يسيراً على المبرمج.
ونذكر في الجدول المرافق توصيفاً لأشهر طرائق الصفّ Array:

الطريقة	توصيف
Clear (Array, Int32, Int32)	إسناد القيم الافتراضية لعناصر المصفوفة، أي استبدال قيم العناصر بالقيمة (0) في حالة الأعداد والقيمة خطأ false في حالة العناصر ذات القيم المنطقية والقيمة null في حالات أخرى
Copy(Array, Array, Int32)	نسخ عناصر مصفوفة إلى مصفوفة أخرى ابتداءً من العنصر الأوّل، ويُحدّد الوسيط الثالث Int32 عدد العناصر
CopyTo(Array, Int32)	نسخ جميع عناصر مصفوفة بدءاً من العنصر الموافق لقيمة المؤشّر المعطاة كوسيط ثاني للطريقة
GetLength(Int32)	تعيد الطريقة طول أحد أبعاد المصفوفة أي عدد عناصره
GetLowerBound(Int32)	تعيد الطريقة دليل العنصر الأوّل في البعد المحدّد للمصفوفة
GetUpperBound(Int32)	تعيد الطريقة دليل العنصر الأخير في البعد المحدّد للمصفوفة
IndexOf(Array, Object)	تعيد قيمة المؤشّر الموافق للغرض المستخدَم كوسيط دخل ثانٍ
Reverse(Array)	عكس ترتيب العناصر في أحد أبعاد المصفوفة
SetValue (Object, Int32)	إدراج عنصر في مكان محدّد من المصفوفة
Sort(Array)	فرز العناصر في أحد أبعاد المصفوفة باستخدام IComparable

مثال:

وفي الرمّاز الآتي، نوضّح استخدام بعض طرائق المصفوفات:

```
using System;
namespace ArrayApp
{

    class MyArray
    {
        static void Main()
        {
            int[] list = { 34, 72, 13, 44, 25, 30, 10 };
            int[] temp = new int[list.Length + 1];
            temp = list;
            temp.SetValue(18, temp.Length - 1);
            Console.Write("Original Array: ");
            foreach (int i in temp)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();

            // reverse the array
            Array.Reverse(temp);
            Console.Write("Reversed Array: ");
            foreach (int i in temp)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
            //sort the array
            Array.Sort(temp);
            Console.Write("Sorted Array: ");
            foreach (int i in temp)
            {
                Console.Write(i + " ");
            }
            Console.WriteLine();
```

```
            Console.ReadKey();
        }// end Main
    }//end class MyArray
}// end namespace ArrayApp
```

نلاحظ أنّ عملية فرز Sort عناصر المصفوفة أو عكس ترتيبها Reverse أو حتى إضافة عنصر إلى الموضع الذي نريد ضمن المصفوفة SetValue أصبحت عمليات سهلة ولا تتطلّب حلقات تكرار، وإنّما فقط استدعاء لطرائق. وبعد تنفيذ الرمّاز السابق نحصل على الخرج الآتي:

```
Original Array: 34 72 13 44 25 30 18
Reversed Array: 18 30 25 44 13 72 34
Sorted Array: 13 18 25 30 34 44 72
```

2. البنية

تُعتبر البنية struct بديلاً قليل التكلفة من ناحية استهلاك الموارد مقارنة بالصفّ. فبينما الصفّ هو نمط معطيات مرجعي Reference Data Type، تُعتبر البنية أحد أنماط القيمة Value Data Type. ويمكن للبنية أن تحتوي: بناة وخصائص وطرائق وحقول ومعاملات ومفهرسات وأنماط متداخلة أخرى، ولا يمكنها احتواء أي هادم Destructor.

ويتمّ استخدام البنية عندما يكون النمط بأغلبيته مصمّماً لتخزين معطيات، ولا يحتوي سوى بعض الطرائق التي تقوم بمهام بسيطة مثل إسناد قيم للخصائص والحقول أو استرجاعها. ويتمّ التعامل مع البنية كأي نمط قيمة آخر، ولا يتطلّب إنشاء متغيّر منها استخدام الكلمة المفتاحية new على الرّغم من أنّه يمكن استخدامها مع بانٍ تمّ تعريفه ضمن البنية. ولا ترث البنية من البنى الأخرى ولا تسمح للبنى الأخرى بوراثتها.

مثال1:

في الرمّاز الآتي، نوضّح البنية Complex المعبّرة عن العدد العقدي:

```
using System;
namespace Structs
{
    public struct Complex
    {
        public int real, imaginary;
        public Complex(int real, int imaginary)
        {
            this.real = real;
            this.imaginary = imaginary;
        }
        public double Magnitude()
        {
            return System.Math.Sqrt(real * real + imaginary * imaginary);
        }
    }// end struct Complex

    class StructTest
    {
        static void Main()
        {
            Complex complexNumber;
            complexNumber.real = 3;
```

```
            complexNumber.imaginary = 4;
            System.Console.WriteLine("Magnitude  of ({0} + i{1}) is: {2}",
   complexNumber.real, complexNumber.imaginary, complexNumber.Magnitude());
        }//end Main
    }// end class StructTest
}// end namespace Structs
```

نلاحظ أنّه تمّت تهيئة حقلي العدد العقدي من خلال إسناد قيمة مباشرة إلى كلّ منهما من دون استدعاء الباني. أي أنّه لإنشاء متغيّر من نمط البنية ليس بالضرورة استخدام الكلمة المفتاحية new. وبعد تنفيذ الرمّاز السابق، نحصل على الخرج الآتي:

```
Magnitude  of (3 + i4) is: 5
```

مثال2:

في الرماز الآتي، نوضّح كيفية تمرير البنية كقيمة إلى طريقة، فلا تتغيّر قيم حقولها قبل استدعاء الطريقة وبعد استدعاء الطريقة على الرّغم من قيام الطريقة بتعديل قيمهم:

```
using System;
namespace Locations
{
    struct Location
    {
        public int xVal;
        public int yVal;
        public Location(int xCoordinate, int yCoordinate)
        {
            xVal = xCoordinate;
            yVal = yCoordinate;
        }// end constructor
        public override string ToString()
        {
            return (String.Format("{0}, {1}", xVal, yVal));
        }
    }// end struct Location

    class Tester
    {
        public void myFunc(Location loc)
```

```
        {
            Console.WriteLine("In MyFunc, before loc: {0}", loc);
            loc.xVal = 50;
            loc.yVal = 100;
            Console.WriteLine("In MyFunc, after loc: {0}", loc);
        }
        static void Main()
        {
            Location loc1 = new Location(200, 300);
            Console.WriteLine("Loc1 location: {0}", loc1);
            Tester t = new Tester();
            t.myFunc(loc1);
            Console.WriteLine("Loc1 location: {0}", loc1);
        }// end Main
    }// end class Tester
}// end namespace Locations
```

نلاحظ أنّه تمّ إنشاء الغرض loc1 ذي الإحداثيات (200,300) من البنية Location باستخدام الباني مسبوقاً بالكلمة new، ثمّ تمّت طباعة إحداثيات loc1 وبعدها تمّ استدعاء الطريقة t.myFunc مع البنية loc1. تقوم الطريقة السابقة بطباعة إحداثيات البنية، ثمّ تستبدل إحداثياتها الأساسية بالإحداثيات (50,100) ثمّ تعيد طباعة الإحداثيات الجديدة. وبعد انتهاء عمل الطريقة، تتمّ طباعة إحداثيات loc1 مرّة أخرى فنحصل على الخرج الآتي:

```
Loc1 location: 200, 300
In MyFunc, before loc: 200, 300
In MyFunc, after loc: 50, 100
Loc1 location: 200, 300
```

نلاحظ أنّ إحداثيات loc1 لم تتغيّر بعد استدعاء الطريقة t.myFunc وذلك لأنّ البنية عبارة عن نمط قيمة وليس نمط مرجع.

1.2. البنية DateTime

تُعدّ البنية DateTime من البنى الشهيرة المسبقة التعريف في C#، وتحتوي على الكثير من الخصائص والطرائق المخصّصة للتعامل مع الوقت والتاريخ.

مثال:

يوضّح الرمّاز الآتي بعض أعضاء هذه البنية:

```
using System;
namespace DateTimeExamples
{
    class DateTimeExample
    {
        static void Main()
        {
            //DateTime Constructor
            DateTime date = new DateTime(2020, 07, 16);
            Console.WriteLine(date.ToString()); // 16-Jul-20 12:00:00 AM
            //current Date and Time
            Console.WriteLine("The current date is: " + DateTime.Now);
            //current Day of Week
            Console.WriteLine("Today is: " + date.DayOfWeek);//Thursday
                                                              //DateTime
Formatting
            Console.WriteLine(date.ToString("MMMM dd, yyyy"));//July 16, 2020
            Console.WriteLine(date.ToString("dd-MM-yyyy"));//16-07-2020
            //// areEqual gets false.
            DateTime date2 = new DateTime(2019, 07, 16);
            bool areEqual = (date == date2);
            if (areEqual)
                Console.WriteLine("The dates are equal");
            else
                Console.WriteLine("The dates are not equal");
        }// end Main
    }// end class DateTimeExample
}// end namespace DateTimeExamples
```

وبعد التنفيذ، يظهر الخرج الآتي:

```
16-Jul-20 12:00:00 AM
The current date is: 16-Jul-20 8:49:43 PM
Today is: Thursday
July 16, 2020
16-07-2020
The dates are not equal
```

3. الصندقة وفكّ الصندقة

ترث البنى والأنماط البسيطة من الصفّ ValueType الموجود في فضاء الأسماء System. ويرث الصفّ ValueType من الصفّ Object. ولهذا فإنّه يمكن إسناد أي قيمة من نمط بسيط إلى متغيّر من النمط Object، تُسمّى هذه العملية بالصندقة Boxing، وتسمح بتغليف نمط بسيط بغرض، وبذلك يمكن استخدام الأنماط البسيطة في أي مكان يتطلّب التعامل مع أغراض. وتتمّ الصندقة من خلال نسخ القيمة إلى غرض ممّا يسمح باستخدام القيمة كغرض، ويمكن القيام بهذه العملية بشكل ضمني أو بشكل صريح كما يبيّن الرمّاز في المثال. إن عملية الصندقة مكلفة من حيث استهلاك الموارد لأنّه يجب إنشاء غرض وحجز مكان في الذاكرة له. ويوجد العديد من بنى المعطيات المفيدة المبرمج، وسنستعرض بعضها في الفصل القادم.

مثال:

```
using System;
class TestBoxing
{
    static void Main()
    {
        int i = 456;

        // Boxing copies the value of i into object o.
        //  object o = (object)i;     //explicit boxing
        object o = i;  // implicit boxing
        object os = "Hello";
        // Change the value of i.
        //  i = 789;

        // The change in i doesn't affect the value stored in o.
        System.Console.WriteLine("The value-type value i= {0}", i);
        System.Console.WriteLine("The object-type value o= {0}", o);

        try
        {
            string j = (string)os;  // attempt to unbox
            System.Console.WriteLine("Unboxing OK.");
        }
        catch (System.InvalidCastException e)
        {
```

```
            System.Console.WriteLine("{0} Error: Incorrect unboxing.",
e.Message);
        }
        Console.ReadKey();
    }//end Main
}// end class TestBoxing
```

وبعد التنفيذ، نحصل على الخرج الآتي:

```
The value-type value i= 456
The object-type value o= 456
Unboxing OK.
```

يتّضح لنا من الرمّاز السابق أنّ لنمط القيمة الأساسي مكان في الذاكرة مختلف عن مكان الذاكرة المخصّص للغرض المغلِّف له، ولذلك كانت القيمتان المخزّنتان فيهما مختلفتين. ومن أجل أن تتمّ عملية فكّ الصندقة بنجاح، يجب أن يتمّ إسناد القيمة الناتجة عن فكّ الصندقة إلى متغيّر من نفس نمط القيمة التي تمّت صندقتها. وإلّا سيتمّ قذف استثناء من النمط InvalidCastException. وعند محاولة فكّ الصندقة عن القيمة null سيتمّ قذف استثناء أيضاً من النمط NullReferenceException.

الأنشطة المرافقة

التمرين الأوّل:

عرّف البنية Point وضمنها العناصر الآتية:

- الخصائص X, Y, Z وجميعها من النمط int وتقبل قيمها بين (0) و(100).
- بانٍ يأخذ ثلاثة قيم كوسائط دخل ويُسندها إلى الخصائص الثلاث.
- بانٍ افتراضي يسند القيمة (0) إلى الخصائص الثلاث.
- الطريقة Amplitude التي ترجع مطال العنصر(الجذر التربيعي للقيمة المطلقة لمجموع مربّعات قيم الخصائص الثلاث).

قم بإنشاء الصفّ Tester الذي يحوي الطريقة Main من أجل اختبار البنية السابقة.

التمرين الثاني:

ضمن الطريقة Main من الصفّ Tester الذي أنشأته في التمرين السابق، عرّف المصفوفة PointArray والتي عناصرها من النمط Point ثم أضف إليها عشرة عناصر. ثمّ عرّف الطريقة SearchMaxAmp التي تبحث ضمن عناصر المصفوفة وتعيد العنصر ذي المطال الأكبر. وأخيراً، قم باختبار ما قمت بتعريفه.

المراجع

1. https://docs.microsoft.com/en-us/dotnet/csharp/.
2. "التصميم والبرمجة غرضية التوجّه"، الدكتور سامي خيمي، الإجازة في تقانة المعلومات، من منشورات الجامعة الافتراضية السورية، الجمهورية العربية السورية، 2018.