الدرس التاسع : تمرير المعاملات بعمق

كنا قد أخذنا موضوع تمرير المعاملات في الوسائل بشكل سريع ، وفي هذا الدرس سوف نتعمق في هذا الموضوع و لكن قبل ذلك أود أن أوضح نقطة مهمة ألا و هي أنواع المتغيرات .

أنواع المتغيرات :

ربما أشرنا فيما سبق من الدروس أن هناك نوعان من المتغيرات نوع ذو قيمة و أخر ذو مرجع .

النوع الأول : المتغيرات القيمية

و هذا النوع يحوي على الأنواع التالية من البيانات :

1 – الأعداد الصحيحة (byte,sbyte,int,uint,long,ulong,char) .

2 – الأعداد ذات النقطة العائمة (double,float) .

3 – الأعداد العشرية (decimal)

4 – المتغيرات المنطقية (true,false)

5 – السجلات structure.

6 – التعدديات Enumeration

وهي تحوي قيمة المتغير فقط ، بحيث إذا قمت بإسناد قيمة متغير إلى آخر فإن قيمته هي التي يتم نسخها ، و طريقة استعمال هذا نوع من المتغيرات هي التي نستعملها من أول درس في هذه الدورة .

النوع الثاني : المتغيرات المرجعية

وتشتمل الأنواع التالية من البيانات :

1 – الأصناف class و منها السلاسل الحرفية string و المصفوفات array .

2 – الواجهات interface

3 – الوكيل delegate .

و تحوي هذا النوع من المتغيرات على مرجع للكائن الذي يمثله هذا المتغير ، فمثلا قبل أن تستخدم صنف يجب عليك تخليقها أولا بواسطة المعامل new – كما في الدرس السابق – ثم ينشئ مراجع إلى ذلك الكائن

خذ المثال التالي :

```
Myclass   x =  new Myclass();
Myclass y;
y = x;
```

قمنا أولا بإعلان عن متغير مرجعي و إسناده إلى كائن الذي أنشأناه بواسطة الكلمة **new**، و في السطر الثاني قمنا بإعلان عن متغير مرجعي و لكن لا يحمل أي مرجع إلى كائن ، في السطر الثالث قمنا بإسناد قيمة المرجع **x** إلى **y** بحيث أصبحا يشيران إلى نفس الكائن و ليس كائنين منفصلين .

تمرير المعاملات بواسطة قيمتها :

و ذلك حسب ما قمنا به في الدرس السابق ، حيث يقوم المترجم بإرسال نسخة من المتغيرات و ليست المتغيرات بنفسها ، و نتيجة ذلك أنه إذا حصلت أي تغيرات على تلك النسخ فإن المتغيرات الأصلية لن تتأثر خذ المثال التالي :

```
using System;

class PassParmater
{
	static void Main()
	{
		PassParmater Example = new PassParmater();
		int x = 6, y = 5;
		Console.WriteLine("x = "+ x + "\t y = " + y);
		Example.ByValue(x,y);
		Console.WriteLine("After Passing Parmater ");
		Console.WriteLine("x = "+ x + "\t y = " + y);

	}
	void ByValue(int x, int y)
	{
		x = 0;
		y = 1;
	}
}
```

الناتج :

```
x = 6    y = 5
After Passing Parmater
x = 6    y = 5
Press any key to continue_
```

بشكل افتراضي يتم تمرير المعاملات سواء أكانت قيمية أم مرجعية بواسطة قيمتها (أي مجرد إرسال نسخ منها) .

حتى الآن الجزء الخاص بالمتغيرات القيمية قد تم شرحه ، أما بالنسبة للمتغيرات المرجعية فإنه يتم إرسال نسخة من المرجع و ليس الكائن ، وعلى العكس فإذا حدث أي تغير على المتغيرات في الوسيلة فإنه سوف يؤثر على الكائن و لكن لن يؤثر على المرجع نفسه .

قد يكون الأمر مربك قليلا ، و لكن تذكر أن المتغيرات المرجعية هي مجرد مؤشرات أو مراجع إلى كائنات مستقلة ، سنأخذ أمثلة على تمرير الأنواع المرجعية في الدروس القادمة بإذن الله تعالى .

إرجاع أكثر من قيمة

مما يعاب على الطريقة الأولى (أي طريقة تمرير المعاملات) هي أنها لا ترجع إلا قيمة واحدة فقط مما يؤدي إلى فصل الوسيلة المترابطة إلى وسيلتين فلهذا ، قرر مخترعو لغة السي شارب اختراع طريقة لإرجاع أكثر من قيمة في وسيلة .

على العموم تعتمد هذه الطريقة على ما يسمى تمرير المعاملات بواسطة مراجعها ، لتبسيط هذه العملية دعنا نأخذ متغير قيمي و نمرره بواسطة مراجعه و نقارن النتيجة، انظر الجدول التالي :

متغير قيمي	ماذا يمرر ؟	هل يتأثر المتغير الأصلي؟
بواسطة قيمتها	نسخة من المتغير	لا
بواسطة مرجعها	المتغير نفسه	نعم

و قبل أن نأخذ مثال على كيفية تمرير المعاملات بواسطة مراجعها يجب علينا أن ننوه أن هناك كلمتان محجوزتان لتمرير المعاملات بواسطة مراجعها ألا و هما **ref** و **out** و الفرق بينهما أن **out** لا تحتاج أن يكون المتغير مسند إليه قيمة معينة بعد الإعلان عنه أما **ref** فهي تحتاج إلى ذلك ، و الآن إلى المثال :

```
using System;
class PassByRef
{
      static void Main()
      {
            PassByRef exmple = new PassByRef();
            int Number1, Number2;
            int sum = 0; // Refلكي نمررها بواسطة
            exmple.GetValue(out Number1, out Number2);
            exmple.Sum(ref sum,Number1,Number2);
            Console.WriteLine("Sum = " + sum);
      }
      void GetValue(out int x, out int y)
      {
            Console.Write("Enter number 1 : ");
            x = Int32.Parse(Console.ReadLine());
            Console.Write("Enter number 2 : ");
            y = Int32.Parse(Console.ReadLine());
    }
      void Sum(ref int sum, int x, int y)
      {
            sum = x + y;
      }
}
```

الناتج :

```
Enter number 1 : 5
Enter number 2 : 4
Sum = 9
Press any key to continue
```

هذا بخصوص المتغيرات القيمية أما المتغيرات المرجعية فالجدول التالي يشرح ماذا يحدث في حالة تمريرها بواسطة قيمتها و مراجعها :

متغيرات مرجعية	ماذا يمرر ؟	ماذا يحدث؟
بواسطة قيمتها	نسخة من المرجع	يتأثر الكائن فقط دون المرجع
بواسطة مرجعها	المرجع نفسه	يتأثر المرجع و الكائن ، و لكن إذا تغير المرجع فإننا سوف نفقد الكائن

قد تريد بعض الأمثلة على المتغيرات المرجعية عندما تمرر بواسطة قيمتها و مراجعها ، و لكن أستميحك أن تؤجل هذا حتى الدرس القادم عند دراستنا للمصفوفات ، سوف نتعرض لهذه الأمثلة .

تعدد التعريفات للوسيلة :

في كثير من الأحيان تريد أن تستعمل وسيلة معينة لأداء مهمة محددة ولكن يجب أن يتخلف نوع المعاملات في كل مرة و ربما عددها .

و لكي أقرب لك هذا المفهوم تصور أنك تريد أن تنشئ وسيلة تقوم بعملية جمع لثلاث متغيرات ، يمكنك أن تنشئ وسيلة للمتغيرات **int** و أخرى **double** بنفس الاسم .

و هذا هو المثال مكتوب لهذه الوسيلة :

```
using System;
class AddThree
{
	static void Main()
	{
		AddThree exmple = new AddThree();
		int x1 = 2, x2 = 3, x3 = 5;
		double y1= 3.0, y2 = 6.0, y3 = 4.0;
		Console.WriteLine(exmple.Add(x1,x2,x3));
		Console.WriteLine(exmple.Add(y1,y2,y3));
	}
	int Add (int x, int y, int z)
	{
		return x + y + z;
	}
	double Add (double x, double y, double z)
	{
		return x + y + z;
	}
}
```

الناتج :

```
10
13
Press any key to continue
```

ترى بوضوح كيف أننا قمنا بإعادة تعريف الوسيلة **Add** مرة للمتغيرات من نوع **int** و مرة من نوع **double** ، و يمكنك أن تزيد و أن ننقص من عدد المعاملات و تغير نوع الراجع كما تريد .

قد تكون هذه الطريقة فيها شيء من الرتابة و التكرار و قد تم تلافي هذه المشكلة في الإصدار الثاني من لغة السي شارب بما يسمى التعميم ، و هو ليس في نطاق هذه الدروس .

عمر المتغيرات و مجالها :

يعرف عمر المتغير بأنه المدة التي يوجد فيها المتغير في الذاكرة ، فالمتغيرات التي توجد في الوسيلة يبدأ عمرها عندما يبدأ المترجم في تنفيذ الجمل التي نعلن عنها وتنتهي عندما ينتهي المترجم من تنفيذ تلك الوسيلة ، تسمى هذه المتغيرات بالمتغيرات ذات العمر الآلي ، أي أنها تبدأ عندما يصل البرنامج إلى مكان إعلانها و تبقى ما دامت القطعة البرمجية التي توجد فيها (صنف ، وسيلة ، حلقة تكرارية) نشطة ، و تنتهي بانتهاء تلك القطعة .

و يوجد هناك نوع ثان من المتغيرات تسمى متغيرات عمر ثابت ، و هذه المتغيرات تبقى في الذاكرة بعد أن يتم الإعلان عنها و لها كلمة محفوظة و هي **static** و لنا لهذه الكلمة عودة بإذن الله تعالى .

مجال المتغيرات :

و هو مدى الذي يمكن من خلاله أن تصل إلى المتغير ، و هذه الخاصية مرتبطة بالتي قبلها ارتباط وثيق ، و لكن الفرق الوحيد أن ليس كل متغير موجود يمكن الوصول إليه و لكن كل متغير يمكن الوصول إليه فهو موجود .

على العموم و كقاعدة فإن مجال المتغير هو القطعة البرمجية التي تحويه (الصنف ، الوسيلة ، الحلقة التكرارية) و التي تبدأ بـ { و تنتهي بـ } ، و القاعدة الثانية أنه إذا كان هناك متغيران بنفس الاسم في الصنف و الوسيلة فإن البرنامج يأخذ الذي يوجد في نفس القطعة البرمجية ، و المثال التالي يوضح ذلك :

```
using System;
class Scope
{
      int x = 1;
      void MethodA()
      {
            int x = 25;
            Console.WriteLine("Local x in MethodA is " + x
                  + " After Entering MethodA");
            ++ x;
            Console.WriteLine("Local x in MethodA is " + x
```

```
                    + " before exiting MethodA");
        }
        void MethodB()
        {
              Console.WriteLine("Instance variable x is " + x
                    + " on entering MethodB");
              x = 10;
              Console.WriteLine("Instance variable x is " + x
                    + " before exiting MethodB");
        }
        static void Main()
        {
              Scope example = new Scope();
              int x = 5;
              Console.WriteLine("Local x in Main is " + x);
              example.MethodA();
              example.MethodB();
              example.MethodA();
              example.MethodB();
        }// end Main
}// end Scope class
```

المخرجات :

```
Local x in Main is 5
Local x in MethodA is 25 After Entering MethodA
Local x in MethodA is 26 before exiting MethodA
Instance variable x is 1 on entering MethodB
Instance variable x is 10 before exiting MethodB
Local x in MethodA is 25 After Entering MethodA
Local x in MethodA is 26 before exiting MethodA
Instance variable x is 10 on entering MethodB
Instance variable x is 10 before exiting MethodB
Press any key to continue
```

ترى أنه عند تنفيذ هذا البرنامج فإن الوسيلة **Main** و **MethodA** تستخدم المتغير المحلي (الموجود في الوسيلة نفسها) **x** بدلا من المتغير العام الذي يوجد في الصنف، و كذلك تلاحظ أن المتغيرات العامة يمكن أن تستخدم في أي مكان في الصنف ليس مثل المتغيرات المحلية التي تنتهي بانتهاء الوسيلة .

التكرار العودي :

يقصد بالتكرار العودي بأن الوسيلة تستطيع مناداة نفسها ، الممتع في الأمر أن هذه الخاصية تسهل الكثير من العبء على المبرمجين من عشاق الرياضيات ، و لتتضح الصورة أكثر خذ مثال مضروب العدد ! فمثلا مضروب خمسة هو :

```
5! = 5* 4*3*2*1
4! = 4*3*2*1
3! = 3*2*1
2! = 2*1
1! = 1
```

بطريقة الحلقة التكرارية يمكنك حل 5! كالتالي :

```
factorial = 1;
for ( int counter = 5; counter >= 1; counter --)
      factorial *= counter
```

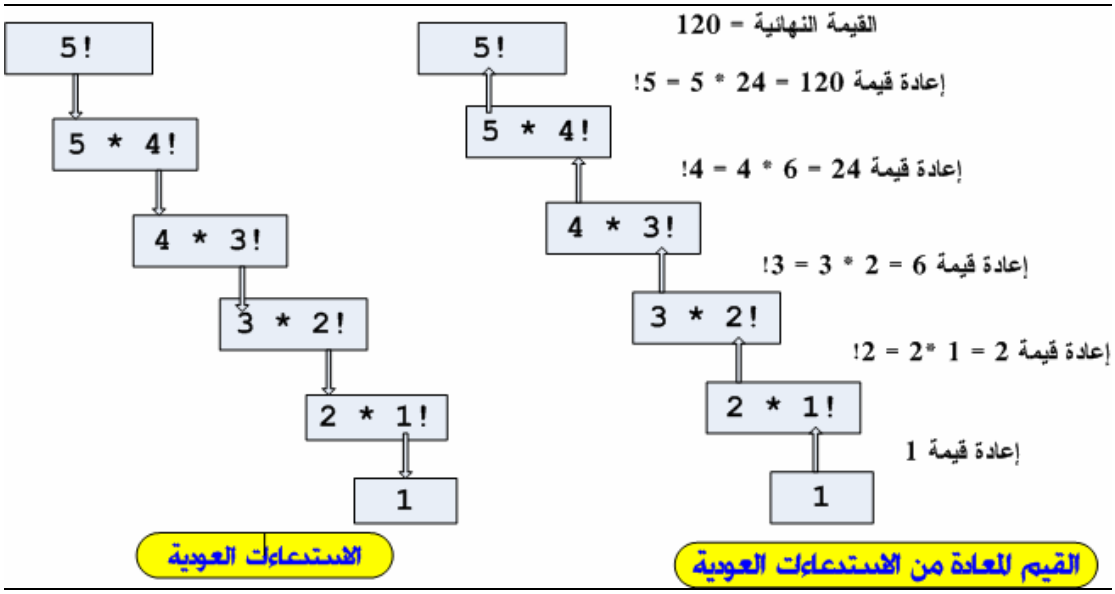
أما بطريقة التكرار العودي فقانون المضروب في الرياضيات يقول التالي :

$$n! = n(n-1)!$$

```
long factorial(long number)
{
      if ( number <= 1) //الأساسية الحالة
            return 1;
      else
            return number * factorial(number -1);
}
```

وحتى لا يسير التكرار العودي إلى ما لا نهاية يجب أن يكون له شرط و هو الحالة الأساسية ، ففي حالة المضروب كانت الحالة الأساسية هي أن مضروب 1 يساوي 1 .

يقوم البرنامج في التكرار العودي بحفظ نسخة من متغيرات في كل مرة يقوم باستدعاء الوسيلة ، فمثلا في مضروب خمسة يقوم أولا بحفظ الرقم خمسة ، و يستدعي المضروب 4 ، و في مضروب أربعة يقوم بحفظ الرقم 4 ، و يستدعي مضروب ثلاثة و هكذا حتى يصل إلى مضروب واحد وهو واحد حيث تقوم الوسيلة بإرجاع واحد، بعد ذلك يقوم البرنامج بحساب مضروب اثنان و من ثم ثلاثة حتى الخمسة مستخدما المتغيرات التي حفظها سابقا ، الشكل التالي يوضح هذه العملية :



قد تكون سهولة حل التكرار حل التكرار العودي هو شفيع لبقائها فهي تأكل الذاكرة أكلا ، و لا تفضل في العمليات الرياضية المعقدة و لكن في المعادلات غير المعقدة لا بأس بها .

الواجب :

1- نقول عن عدد صحيح أنه عدد تام Perfect number إذا كان مجموع عوامله الأولية factors بما فيها العدد 1 (ما عدا العدد نفسه) ساوي العدد ذاته ، على سبيل المثال ، لدينا 6 عبارة عن عدد تام لأن 6 = 1 +2 +3 ، اكتب وسيلة تستقبل عدد ثم تحدد إذا كان تاما أم لا ، استخدم هذه الوسيلة في برنامج يقوم بتحديد و طباعة كافة الأعداد التامة المحصورة بين العددين 1 و 1000 ، أطبع أيضا العوامل الأولية لكل عدد تام و تأكد أن مجموع هذه العوامل يساوي العدد نفسه .

2- نقول عن عدد صحيح انه أولي prime إذا كان لا يقبل القسمة إلا على نفسه و على الواحد ، على سبيل المثال ، لدينا 2 ، 3، 5 ، 7 مجوعة أعداد أولية ، لكن الأعداد 4 ، 6 ، 8 ، 9 ، ليست أعداد أولية :

أ- اكتب وسيلة تحدد فيما إذا كان عددا ما أوليا أم لا .

ب-استخدم الوسيلة السابق ضمن برنامج لتحديد و طباعة جميع ا لأعداد الأولية التي بين العددين 1 و 10000 .

3- نسمي أكبر عدد صحيح قادر على قسمة عددين صحيحين بدون باقي بالقاسم المشترك الأعظم ، اكتب وسيلة عودية ، تستقبل عددين ، و ترجع القاسم المشترك الأعظم ، بحيث إذا كان العددين هما س وص في القاسم المشترك الأعظم يحدد بالتالي : إذا كانت ص تساوي صفر فإن القاسم يساوي س ، و إلا فإن الوسيلة (س ، ص) تساوي الوسيلة (ص ، س ٪ ص) حيث ٪ هي باقي القسمة الصحيحة .

4- اكتب وسيلة عودية تقوم بحساب سلسلة الفبونك Fibonacci و هي تبدأ كالتالي :

0, 1, 1 , 2 , 3 , 5 , 8 , 13 , 21 ...

و يكمن تعريف هذه السلاسل عوديا كالتالي :

```
Fibonacci ( 0) = 0
Fibonacci ( 1) = 1
Fibonacci ( n) = Fibonacci ( n -1)  + Fibonacci ( n - 2)
```

اكتب برنامج يقوم باستقبال عدد n و يظهر بعد ذلك قيمته في السلسلة .

```
*****
***
*
```