

الباب الثالث

أنماط البيانات

Data Types

## محتويات الباب

- أنماط لغة سي#
- الأنماط المبنية في اللغة (Built-in Types)
- أنماط القيمة وأنماط المرجع (Value and Reference Types)
- الصندوق وإلغاء الصندوق (Boxing and Unboxing)
- الأنماط البسيطة (Simple Types)
- المؤثرات والتعبيرات الحسابية
- الأنماط الصحيحة والحقيقية
- تحويل الأنماط (Conversion)
- اللبئات (Characters)
- الحرفيات (Strings)
- فورمات النتائج (Formatting Results)
- قراءة المدخلات من لوحة الأزرار (Keyboard Input)

## أنماط لغة سي#

تتقسم الأنماط بصفة عامة في لغة سي# إلى نوعين:

- أنماط القيمة (Value types): وهي أنماط المتغيرات التي تحتزن قيم البيانات نفسها.
- أنماط المرجع (Reference types): وهي تمثل المتغيرات التي تحتزن عنوان الذاكرة الذي يشير إلى البيان الموجود في حيز آخر من الذاكرة.

كما يوجد نوع ثالث وهو نمط المؤشرات (Pointers) الذي يستخدم في كتابة الكود غير المأمون (Unsafe code) ، ولكننا لن نتعرض له في هذا الكتاب فهو يخرج عن صلب لغة سي# ، وقد أضيف إلى اللغة بغرض التوافق مع سي++.

### الأنماط المبنية في اللغة

إن أنماط سي# المبنية في اللغة ما هي إلا أسماء مستعارة لبعض الأنماط المعروفة في المعمار دوت.نت ، وهي جميعاً موضحة بالجدول التالي حيث نرى اسم النمط وأمامه الاسم المؤهل للنمط المعرف في الحيز System بمكتبة فصول دوت.نت.

الباب الثالث (أنماط البيانات)

جدول (3-1) الأنماط المبنية في لغة سي#

نمط سي#	نمط المعمار دوت.نت	اسم النمط
bool	System.Boolean	البولياني
byte	System.Byte	البايت
sbyte	System.SByte	البايت القصيرة
char	System.Char	اللبنة
decimal	System.Decimal	العشري
double	System.Double	مضاعف الدقة
float	System.Single	الحقيقي
int	System.Int32	الصحيح
uint	System.UInt32	الصحيح بدون إشارة
long	System.Int64	الطويل
ulong	System.UInt64	الطويل بدون إشارة
object	System.Object	الهدف*
short	System.Int16	القصير
ushort	System.UInt16	القصير بدون إشارة
string	System.String	الحرفي

\* لاحظ أن كلمة الهدف هنا هي اسم فصيلة من فصائل دوت.نت أما الأهداف (أو الأمثلة)

التي تنشئها في برامجك فهي مختلفة.

ومن الجدير بالذكر أنه يتساوى أن تستخدم أنماط دوت.نت أو أنماط سي

# فى برنامجك ، أى أن العبارتين التاليتين متساويتان تماماً فكلاهما تعلن عن متغير صحيح وتخصص له القيمة 25:

```
int myInt = 25;  
System.Int32 myInt = 25;
```

كذلك تتساوى كل من العبارتين التاليتين فى الإعلان عن متغير من النمط object:

```
object myObject;  
System.Object myObject;
```

كما أن الأنماط فى هذا الجدول فيما عدا النمط string والنمط object تسمى الأنماط البسيطة (Simple Types).

## خصائص أنماط القيمة (Value Types)

تشمل أنماط القيمة الأنماط العددية (النمط الصحيح والحقيقى إلى آخره) والنمط البولياني (bool). كما تشمل الأنماط المبتكرة مثل المنشأ (struct) ، ومنشأ القائمة (Enumeration) وسوف نتعرف بهذه الأنماط تباعاً. وجميع أنماط القيمة موضحة بالملحق (أ).

وتخزن قيم أنماط القيمة فى حيز الذاكرة المسمى بالخرزنة (Stack).



### الخرزنة (Stack):

يطلق اسم الخرزنة على نطاق الذاكرة المستخدم فى اختزان المتغيرات المحلية وأنماط القيمة.

وتتحدّر جميع أنماط القيمة من فصيلة الهدف "object" وهو جذر شجرة الفصائل فى مكتبة فصول دوت.نت. ومع ذلك فإن أنماط القيمة لا تورث.

## شحن المتغيرات

لا يمكنك استخدام أى متغير محلى بدون شحنه بقيمة ما. والمقصود بالاستخدام هو دخول المتغير فى أى عملية مثل طباعة محتوياته. وتتم عملية شحن المتغيرات كالمثلة الآتية:

```
int myValue = 123;  
int myValue = 0;  
int myValue = new int();
```

فى المثال الأول قمنا بشحن المتغير myValue بالقيمة 123. وفى المثال الثانى شحناه بالقيمة صفر. وفى المثال الثالث حققنا نفس الهدف كما فى المثال الثانى باستخدام الكلمة new التى تستدعى دالة البناء (Constructor) ( لشحن المتغير بالقيمة الابتدائية سابقة التعريف (وهى 0 فى حالة النمط الصحيح int). وهذا يعنى أن العبارتين الآتيتين متكافئتان:

```
int myValue = 0;  
int myValue = new int();
```

وتتميز أنماط القيمة بأن لكل منها دالة بناء تقوم بشحنها بقيمة ابتدائية سابقة التعريف. وهذه القيم موضحة بالملحق (ب).

وفى حالة استخدام الأنماط المبتكرة مثل نمط المنشأ (struct) فإنك عندما تخلق هدفا من المنشأ تستخدم الكلمة new التى تشحن أعضاء المنشأ

تلقائياً بالقيم سابقة التعريف. والآتى بعد مثال لمنشأ يمثل نقطة واقعة عند الإحداثى  $x, y$ :

```
struct Point
{
    int x;
    int y;
}
```

ولإعلان هدف من هذا المنشأ نستخدم عبارة مثل:

```
Point myPoint = new Point();
```

بهذه العبارة فإن جميع عناصر المنشأ ( $x$  و  $y$  فى هذه الحالة) قد تم شحنها بالقيمة 0 وهى القيمة سابقة التعريف. وسوف نقدم المزيد عن المنشآت فى الفقرات القادمة.

## خصائص أنماط المرجع (Reference Types)

أما نمط المرجع فهو يناظر المراجع (References) فى لغة سى++ حيث أن المتغير من نمط المرجع لا يحتوى على البيانات نفسها بل يحتوى على عنوانها. والمتغير المرجع يشغل حيزاً فى الخزانة (Stack) مثل متغيرات القيمة ، أما البيانات نفسها فهى توجد فى حيز آخر من الذاكرة وهو الكوم (Heap).



الكوم (Heap):

يطلق اسم الكوم على حيز الذاكرة المستخدم فى اختزان البيانات التى تشير إليها أنماط المرجع.

والآتى بعد أنماط المرجع المستخدمة فى سى#.

جدول (3-2) أنماط المرجع (Reference Types)

اسم النمط	الكلمة المستخدمة في الإعلان
الفصيلة	class
الوصلة البينية	interface
المندوب	delegate
الهدف (الفصيلة الأساسية في فئات دوت.نت)	object
الحرفي	string

ومن الجائز التحويل ما بين أنماط القيمة وأنماط المرجع باستخدام عملية "الصندوقة" (Boxing) والعملية العكسية "إلغاء الصندوقة" (Unboxing).

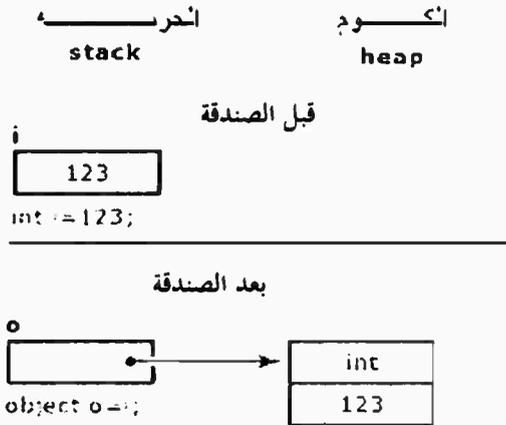
### الصندوقة وإلغاء الصندوقة (Boxing and Unboxing)

تتم عملية الصندوقة بتحويل أي نمط قيمة إلى نمط الهدف (object) كالمثال التالي:

```
int i = 123; // متغير صحيح
```

```
object o = i; // object تخزين المتغير في متغير آخر من النمط
```

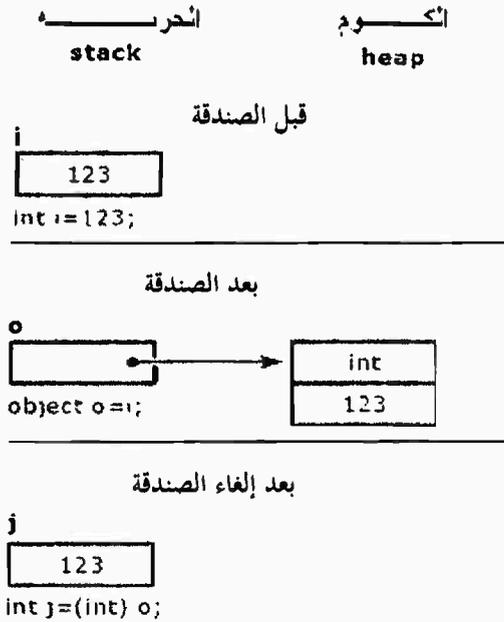
بموجب هذه العبارة يتم اختزان النمط int والقيمة 123 في "الكوم" أما المتغير o فهو مرجع يشير إلى كل منهما. أنظر الشكل التالي:



لتحويل المتغير والبيان المصاحب له إلى نمط قيمة مرة أخرى فإننا نستخدم عملية إلغاء الصندوقة ، وهي عبارة عن تحويل النمط object إلى النمط int (أو أي نمط آخر من أنماط القيمة) باستخدام الإسقاط كما في العبارة التالية:

عملية إلغاء الصندوقة باستخدام الإسقاط (Cast) // `int j = (int) o;`

إن هذه العبارة تخلق متغيراً `j` من نمط القيمة في "الخزنة" ويحتوى على نفس البيان 123. والشكل التالي يوضح العمليات الثلاث الأخيرة:



### مثال (3-1)

```
// Example 3-1.cs
// Boxing and Unboxing

using System;

public class BoxingAndUnboxing
{
    static void Main()
    {
        // Declare a value type :
        int i = 123;
        // Boxing and changing the value :
        object o = i + 456;
        // Unboxing :
        int j = (int) o;
    }
}
```

```
Console.WriteLine("i = {0}", i);  
Console.WriteLine("o = {0}", o);  
Console.WriteLine("j = {0}", j);  
}  
}
```

### تنفيذ البرنامج:

```
i = 123  
o = 579  
j = 579
```

نلاحظ في هذا المثال أننا غيرنا القيمة المخزنة في المتغير المرجع أثناء عملية الصندوق وذلك بإضافة العدد 456 إلى القيمة الأصلية. ومع ذلك فهذا لم يغير من قيمة المتغير الأصلي *i* الذي ظل محتفظاً بقيمته الأصلية بعد الصندوق. أما المتغير *z* فهو يحتوى على نفس قيمة المرجع

.0



بالرغم من أن مفهوم أنماط المرجع لا يختلف عن مفهوم المؤشرات في لغة سي++ ولكن الفارق أنه عند استخدام أنماط المراجع فإن اللغة تعفيك من الدخول في تفاصيل المؤشرات أو متابعة حالة الذاكرة وحذف المؤشرات التي لا تحتاج إليها. إن هذا كله يتم في الخلفية بدون أن تشعر.

## الأنماط البسيطة (Simple Types)

يوضح الجدول التالي الأنماط البسيطة ، وهي تشمل الأنماط العددية (الصحيح "Integral" منها والحقيقي "Floating-point") علاوة على النمط

الباب الثالث (أنماط البيانات)

البولياني "bool" ، ونمط اللبنة "char". ويطلق على هذه الأنماط اسم الأنماط البسيطة لتمييزها عن الأنماط المبتكرة التي يقوم البرنامج بتأليفها مثل المنشأ (struct) والقائمة (enum). ومن البديهي أن جميع الأنماط البسيطة أنماط قيمة.

جدول (3-3) الأنماط البسيطة

اسم النمط	السعة بالبايت	مدى البيان الذي يسمح به
bool	1	يحتوى إما على <sup>(1)</sup> true أو false
byte	1	من 0 إلى 255 (بدون إشارة)
char	2	من U+0000 إلى U+ffff (يونيكود (Unicode) <sup>(2)</sup> )
decimal	16	من $1.0 \times 10^{-28}$ إلى $7.9 \times 10^{28}$
double	8	من $5.0 \times 10^{-324}$ إلى $1.7 \times 10^{308}$
float	4	من $1.5 \times 10^{-45}$ إلى $3.4 \times 10^{38}$
int	4	من -2,147,483,648 إلى 2,147,483,647
long	8	من -9,223,372,036,854,775,808 إلى 9,223,372,036,854,775,807
sbyte	1	من -128 إلى 127
short	2	من -32,768 إلى 32,767
uint	4	من 0 إلى 4,294,967,295 (بدون إشارة)
ulong	8	من 0 إلى

### الباب الثالث (نمط البيانات)

18,446,744,073,709,551,615 (بدون إشارة)		
من 0 إلى 65,535 (بدون إشارة)	2	ushort

#### ملاحظات على الجدول:

(1) لا يوجد تحويل بين بيان من النمط bool والبيان العددي كما في لغة سي++. أى أن القيمة false لا تساوى صفرأ وكذلك القيمة true لا تساوى قيمة غير صفرية.

(2) لاحظ أن اللبنة (char) سعتها 2 بايت وتتسع للحروف المكتوبة بالنظام العالمي الجديد "يونيكود".

#### أمثلة على استخدام الأنماط البسيطة

فيما يلي بعض الإعلانات للأنماط البسيطة:

```
int myInt;  
char aChar;  
long theLongNumber;
```

كما يمكنك الإعلان عن المتغير والتخصيص له في نفس الوقت

باستخدام عبارات مثل:

```
byte myBite = 5;  
short myShort = -223;  
char myChar = 'A';
```

## البيانات العددية (Numeric Data)

قد تسمى البيانات العددية أحياناً بالثوابت العددية وذلك لأن البيانات عامة تختلف عن المتغيرات فى أنها عناصر ثابتة لا يمكن تغييرها ، بينما المتغيرات عبارة عن أوعية للبيانات تتغير محتوياتها بحسب ما تضعه فيها من بيانات. ومع ذلك فسوف نستخدم اسم البيانات (بدلاً من الثوابت) حتى لا يحدث خلط عندما نتحدث عن الثوابت المسماه (أو المتغيرات الثابتة).

### التعبيرات الحسابية

تستطيع فى لغة سى# أن تبني ما تشاء من التعبيرات الحسابية باستخدام البيانات والمتغيرات والمؤثرات.

#### مؤثرات العمليات الحسابية الأساسية

تبني التعبيرات الحسابية باستخدام المؤثرات الحسابية مثل + ، - ، \* ، / وهى المؤثرات الحسابية البسيطة المعروفة فى سائر اللغات لإجراء الجمع والطرح والضرب والقسمة. وعلى سبيل المثال يمكنك كتابة تعبير حسابى مثل:

$$y*(2/3) - 1.0;$$

وعندما تُجرى العمليات الحسابية فإن الضرب والقسمة تكون لهما أولوية قبل الجمع والطرح. ولنعتبر المثال الآتى:

$$6 + 4 * 5;$$

فى هذا التعد فإن المقدار  $4*5$  يتم تقييمه أولاً إلى 20 ، ثم يجمع الناتج على 6 أى .ون النتيجة 26. ويمكنك أن تستخدم الأقواس لتغيير الأولوية. والأقواس يتم تقييمها من الداخل إلى الخارج. ولنعتبر الصورة الآتية للتعبير:

$$(6 + 4) * 5$$

فى هذه الحال يتم تقييم عملية الجمع أولاً (10) ثم يضرب الناتج فى 5 أى أن النتيجة تصبح 50. ومن الجائز أن تخصص تعبيراً بأكمله إلى متغير ، مثل:

$$x = y*(2/3) - 1.0;$$

مؤثرات الزيادة والنقصان

فى لغة سى# نستخدم نفس مؤثرات الزيادة (Increment) والنقصان (Decrement) المستخدمة فى لغة سى++ ، انظر المثال الآتى:

زيادة محتويات المتغير x بمقدار 1 //  $x++;$

إنقاص محتويات المتغير x بمقدار 1 //  $x--;$

وتتوقف قيمة التعبير على مكان المؤثر ++ أو -- (هل هو قبل المتغير أو بعده). فعلى سبيل المثال التعبير:

$$--y + x$$

معناه أن يتم إنقاص قيمة y بمقدار 1 أولاً ثم يضاف الناتج إلى x. أما

التعبير:

$$y-- + x$$

فيعنى إنقاص قيمة  $y$  بمقدار 1 بعد إجراء عملية الجمع. ومن الجائز تخصيص النتيجة لمتغير كالمثال الآتى:

```
y = y-- + x;  
أو  
z = ++y + x
```

### مثال (3-2)

```
// 3-2.cs  
// Increment and Decrement  
using System;  
  
class ArithmeticOperators  
{  
    public static void Main()  
    {  
        int x = 10;  
        int y = 100;  
        int z = y;  
  
        y = y -- + x;  
        Console.WriteLine(y); // النتيجة: 110  
        z = --z + x;  
        Console.WriteLine(z); // النتيجة: 109  
    }  
}
```

### تنفيذ البرنامج:

```
110  
109
```

والمالحق (ج) يوضح أولويات جميع المؤثرات فى لغة سى#. #.

### تدريب (3-1)

ابدأ بالتخصيص للمتغيرات الآتية:

```
int x = 10;
int y = 100;
int z = y;
```

ثم اكتب برنامجاً لحساب وطباعة قيم كل من  $y, z$  بعد إجراء العمليات الآتية:

```
y = y++ + x;
z = ++z + x;
```

### ملاحظة:

لاحظ أن التعبير  $y+++x$  يكافئ تماماً التعبير  $y++ + x$  وذلك لأن الارتباط (Associativity) لمؤثرات الزيادة والنقصان يتم تقييمه من اليمين إلى اليسار بخلاف المؤثرات الأخرى. ومع ذلك فإنه من الأفضل ترك مسافة خالية أو استخدام الأقواس  $(y++) + x$  لتوضيح المعنى (للقارئ).

### كيف تحصل على اسم النمط

للحصول على أنماط المتغيرات أو التعبيرات استخدم أسلوب دوت.نت

بالطريقة الآتية: **GetType()**

```
Console.WriteLine(myVariable.GetType());
```

❖ حيث `myVariable`: المتغير أو التعبير المراد طباعته نمطه.

والنتيجة التي نحصل عليها هي أنماط دوت.نت وليست أنماط

سى#. #.

### تدريب (3-2)

جرب الأمثلة الآتية للحصول على أنماط التعبيرات الموضحة:

```
Console.WriteLine(123.GetType());  
Console.WriteLine(3.14.GetType());
```

وهذه هي النتائج التي تحصل عليها:

```
System.Int32  
System.Double
```

### تقييم التعبيرات ذات الأنماط المختلطة

عندما تكتب تعبيراً حسابياً يشمل أنماطاً مختلفة من البيانات العددية فإن التعبير يقيم كالاتي:

- إذا كان التعبير محتويماً على بيان من النمط double فإن النتيجة تكون من النمط double.
- إذا لم يكن هناك بيان من النمط double فإن النتيجة تكون من النمط float.

### مثال (3-3)

```
// Example 3-3.cs  
// Expressions with mixed types  
  
using System;  
  
class NumbersClass  
{  
    static void Main()  
    {  
        int x = 128;  
        short y = 34;  
        double z = 3.14;  
    }  
}
```

```
// Print the result and the type of result:  
Console.WriteLine("Sum: {0}", x + y + z);  
Console.WriteLine("Type: {0}", (x + y + z).GetType());  
}  
}
```

تنفيذ البرنامج:

```
Sum: 165.14  
Type: System.Double
```

### إضافة لاحقة إلى البيان (Prefixing Numeric Data)

يلزم مع بعض الأنماط أن تضيف حرفاً ما إلى نهاية البيان العددي (لاحقة Suffix) لتمييزه عن بقية الأنماط.

### الأنماط الحقيقية

يعامل المترجم الكسور على أنها من النمط المضاعف الدقة (double) ما لم تميزها بلاحقة ما.

○ مع النمط العشري (decimal) نستخدم الحرف M كالاتي:

```
23.4M;
```

○ مع النمط الحقيقي (float) نستخدم الحرف F كالاتي:

```
23.4F
```

ولو أنك كتبت العدد بدون أي حرف فإنه يعامل على أنه من النمط .double

وعند الإعلان عن أحد هذه الأنماط فإن استخدام اللاحقة ضروري ، وإلا فإن المترجم يرسل لك رسالة بالخطأ ولا تتم الترجمة. وعلى سبيل المثال فإن الإعلانات التالية يعترض عليها المترجم:

```
decimal myAmount = 23.4; // error
```

```
float myRealNumber = 23.4 // error
```

ملاحظة:

بالرغم من أنه يمكنك استخدام الحروف الكبيرة أو الصغيرة لكتابة اللواحق مثل F ، M ، وغيرها ، لكننا سوف نستخدم الحروف الكبيرة في هذا الكتاب.

مثال (3-4)

```
// Example 3-4.cs
// Numeric types

using System;

class MyPoint
{
    static void Main()
    {
        decimal myDc = 23.4M;
        float myFl = 23.4F;
        double myDb = 23.4;

        Console.WriteLine("myDc = {0} \nmyFl = {1} \nmyDb = {2}",
            myDc, myFl, myDb);
    }
}
```

تنفيذ البرنامج:

```
myDc = 23.4
myFl = 23.4
myDb = 23.4
```



هل لاحظت أنك لا تستطيع كتابة الحرفي المراد طباعته على بضعة أسطر؟ إنك تستطيع بدلاً من ذلك أن تستخدم علامة السطر الجديد (\n) بداخل علامتي الاقتباس. والعلامة \n هي نفسها المستخدمة في لغة سي++.

### تدريب (3-3)

اكتب أنماط المتغيرات بالبرنامج السابق باستخدام الأسلوب (GetType()) وسوف تحصل على النتائج الآتية:

```
System.Decimal  
System.Single  
System.Double
```

### الأنماط الصحيحة

أما البيانات الصحيحة مثل int, uint, long, ulong فإنك لا تحتاج معها إلى إضافة حرف إلى نهاية الثابت العددي لأنه عند تقييم التعبير سوف يتم اختيار المخزن المناسب له بحسب قيمة العدد نفسه.

ويمكنك كاختيار أن تستخدم الحروف L ، U ، أو UL لتحديد نوع البيان العددي. وبحسب قيمة العدد يتم تخزينه في المخزن المناسب كالاتي:

○ عند استخدام الحرف L: يخزن العدد على أنه من النمط long أو

ulong بحسب قيمته. وعلى سبيل المثال فإن التعبير الآتي سوف

يعامل على أنه من النمط long لأنه أقل من حدود النمط ulong:

```
4294967296L
```

- عند استخدام الحرف U: يختزن العدد على أنه من النمط long أو uint بحسب قيمته. وعلى سبيل المثال فإن التعبير الآتى سوف يعامل على أنه من النمط uint لأنه أقل من حدود النمط long:  
4294967290U
- عند استخدام الحرفين U و L معاً بأى ترتيب: يختزن العدد على أنه من النمط ulong.

### تدريب (3-4)

جرب الأمثلة الآتية:

```
Console.WriteLine(9223372036854775808L.GetType());  
Console.WriteLine(123UL.GetType());  
Console.WriteLine(4294967296L.GetType());  
Console.WriteLine(4294967290U.GetType());
```

وهذه هي النتائج التي تحصل عليها:

```
System.UInt64  
System.UInt64  
System.Int64  
System.UInt32
```

### تحويل الأنماط (Conversion)

يوجد فى لغة سى# تحويل ضمنى (Implicit) أو تحويل صريح (Explicit) للأنماط. وعلى سبيل المثال ، تستطيع أن تكتب العبارة التالية للإعلان عن متغير حقيقى:

```
double myVar = 4;  
float myVar = 33;
```

تتضمن هذه الإعلانات عملية تحويل من النمط الصحيح إلى الأنماط الحقيقية float و double. ولكن هذه العمليات تتم في الخلفية بدون أن تشعر بها. فأنت تكتب العدد الصحيح 4 أو 33 ، ويتولى المترجم عملية التحويل. ومع ذلك فلا يجوز كتابة العبارة الآتية:

```
float myVar = 3.3; // Compilation error
```

إن هذه العبارة تعنى أنك تريد تخزين العدد 3.3 (وهو من النمط double — كما ذكرنا في الفقرات السابقة) في مخزن أقل من حجمه وهو مخزن النمط float. ولذلك فإن المترجم يعترض على هذه العبارة ولا يتم الترجمة. (يمكنك استخدام اللاحقة F مع العدد كما ذكرنا من قبل ، وهي تؤدي إلى تحويل البيان العددي نفسه).

والقاعدة العامة لتحويل الأنماط تسمح بتخزين نمط أقل حجماً في مخزن أكبر حجماً ، ولكن العكس غير جائز.

ولنعتبر الإعلانات التالية عن متغير من النمط int وآخر من النمط double:

```
int x = 4;
double y = 3.0;
```

إن العبارات التالية تصبح غير جائزة:

```
short z1 = x; // Compilation error
short z2 = y; // Compilation error
```

والسبب في ذلك أنه لا يوجد تحويل ضمنى من النمط int أو double إلى النمط short فهي جميعاً أكبر من مخزن النمط short.

ولإتمام عملية التحويل فإنه لابد من استخدام الإسقاط ، أى:

```
short z1 = (short)x;
```

```
short z2 = (short)y;
```

ولتخزين بيان عددي صحيح في متغير فلا بد من التأكد من سعة المخزن ، وعلى سبيل المثال فإن العبارة التالية تحاول تخزين عدد صحيح (int) في المخزن المخصص للنمط القصير (short) ولكن العدد المراد تخزينه أكبر من سعة المخزن:

```
short x = 32768; // Compilation error
```

ولا يجوز استخدام الإسقاط في مثل هذه الحالة.

كذلك فإن عملية الإسقاط التالية غير جائزة لأنها تتضمن تحويل ضمنى من النمط double إلى النمط float:

```
float z = (double) 4.4; // Compilation error
```

والملاحق (د) و (ه) تحتوي على التحويلات الضمنية والصريحة الممكنة للبيانات العددية.

## البنات (char)

إن اللبنة (character) تعنى الحرف أو الرقم أو الرمز ، وهى تخزن فى صورة يونيكود (Unicode) فى حيز قدره 2 بايت. وإذا كنت جديداً على النظام يونيكود ، فإنه النظام الذى تلا النظام آسكى (ASCII) لتكويد اللبنة. والنظام يونيكود يتسع لتكويد لبنة جميع اللغات فى العالم بما فيها العربية والصينية واليابانية. ويجوز أن تكتب اللبنة بصور مختلفة كالآتى (جميع الأمثلة التالية تمثل الحرف A بالنظم المختلفة):

- اللبنة الحرفية الصريحة مثل:

```
char myChar = 'A'; // لبنة حرفية صريحة
```

- أو رقم سداسي عشر (Hexadecimal) مثل:

```
char myChar = '\x0041'; // رقم سداسي عشر
```

لاحظ ان الرقم المكتوب بالكود السداسي عشر يأتي مسبقاً بالعلامة \x.

- أو رقم يونيكود مثل:

```
char myChar = '\u0041'; // يونيكود
```

لاحظ ان الرقم المكتوب باليونيكود يأتي مسبقاً بالعلامة \u.

- أو بتحويل الكود آسكى باستخدام الإسقاط مثل:

```
char myChar = (char)65; // تحويل العدد 65 المثل للحرف A
```

وفي جميع الأحوال نستخدم علامتى الاقتباس المفردة لاحتواء اللبنة.

يوضح المثال التالى طريقتين لتخزين اللبنة 'A' باستخدام الكود العدى للبنة ، وكذلك طريقة طباعتها.

مثال (3-5)

```
// Example 3-5.cs
// char conversion

using System;

class CharClass
{
    static void Main()
    {
```

```
char myChar = (char)65;
int yourChar = 'A';

// Print the result:
Console.WriteLine("The character is: {0}", myChar);
Console.WriteLine("The code of the character is: {0}",
    yourChar);
}
```

### تنفيذ البرنامج:

```
The character is: A
The code of the character is: 65
```

### ملاحظات على البرنامج السابق:

لاحظ أنه من الممكن كتابة تعبير اللبنة بالصورة:

```
int yourChar = 'A';
```

ومعنى ذلك أنه يوجد تحويل ضمنى من النمط char إلى النمط int ، ولكن العكس غير جائز. أى أن العبارة التالية غير جائزة بدون إسقاط:

```
char myChar = 65; // Compilation error
```

وهذه هي نفس العبارة باستخدام الإسقاط:

```
char myChar = (char)65;
```

### **فورمات النتائج (Formatting Results)**

بعد أن مررنا بجميع أنواع البيانات العددية ، لعلك تنتظر أن ترى طريقة مناسبة لطباعة النتائج ، فقد تستلزم بعض التطبيقات كتابة الناتج العددي فى خانتي عشريتين ، كما تتطلب بعض التطبيقات كتابة علامة النقود (الدولار أو الجنيه) بجانب النتيجة. وهناك طريقة لإجراء

الفورمات تستخدم مع أسلوب الطباعة Write أو WriteLine نقدمها في هذه الفقرة. كما أن الملحق "و" يلخص عمليات الفورمات في جدول يمكنك استخدامه كمرجع سريع.

### فورمات العملة (Currency)

لطباعة العدد مصحوباً بعلامة العملة مع إضافة الأصفار اللازمة في الخانات العشرية استخدم لبنة الفورمات C أو c كالمثال الآتي:

```
Console.WriteLine("{0:C}", 1.2);
```

إن العدد 1.2 يظهر في الخرج بالصورة \$1.20.

ولو كان البيان العددي سالبا (-1.2) كما في المثال التالي:

```
Console.WriteLine("{0:C}", -1.2);
```

فإنه سوف يظهر بين قوسين بالصورة (\$1.20).

### الفورمات العشرية (Decimal)

لطباعة العدد مسبوفاً بعدد معين من الأصفار استخدم لبنة الفورمات D أو d كالمثال الآتي:

```
Console.WriteLine("{0:D5}", 123);
```

ونتيجة هذه العبارة هي 00123. أي أن عدد الأرقام في النتيجة يحددها الرقم الذي على يمين الحرف D.

### فورمات العلامة العشرية الثابتة (Fixed-point)

لطباعة العدد محتوياً على عدد معين من الخانات العشرية استخدم لبنة الفورمات F أو f كالأمتثلة الآتية:

```
Console.WriteLine("{0:F2}", 12); // خانتان  
Console.WriteLine("{0:F0}", 12); // بدون خانات عشرية  
Console.WriteLine("{0:F0}", 12.3); // حذف الكسور العشرية  
Console.WriteLine("{0:F2}", 12.3); // خانتان
```

تنتج هذه العبارات النتائج الآتية بالترتيب:

```
12.00  
12  
12  
12.30
```

### الفورمات العامة (General)

لطباعة العدد بالصورة سابقة التعريف استخدم لبنة الفورمات G أو g  
كالأمثلة الآتية:

```
Console.WriteLine("{0:G}", 1.2);  
Console.WriteLine("{0:G}", 1.23);
```

تنتج هذه العبارات النتائج الآتية بالترتيب:

```
1.2  
1.23
```

وهي نفس النتائج التي نحصل عليها باستخدام فورمات على الإطلاق ،  
مثل:

```
Console.WriteLine("{0}", 1.2);  
Console.WriteLine("{0}", 1.23);
```

### الفورمات العددية (Numeric)

لطباعة العدد محتوياً على فواصل بين الأرقام وعلامتان عشريتان  
استخدم لبنة الفورمات N أو n كالمثال الآتي:

```
Console.WriteLine("{0:N}", 1230000000);
```

ونتيجة هذه العبارة هي العدد:

1,230,000,000.00

### الفورمات العلمية (Scientific)

لطباعة العدد بالصورة الأسية المستخدمة مع الأرقام الكبيرة ولا سيما

في التطبيقات العلمية ، استخدم لبننة الفورمات E أو e كالمثال الآتي:  
`Console.WriteLine("{0:E}", 12300000);`

ونتيجة هذه العبارة هي العدد:

1.230000E+007

### الفورمات السداسى عشر (Hexadecimal)

لطباعة العدد بالكود السداسى عشر استخدم لبننة الفورمات X أو x

كالأمثلة الآتية:

```
Console.WriteLine ("{0:X}", 123);  
Console.WriteLine ("{0:X}", 65535);
```

ونتيجة هذه العبارات بالترتيب هي:

7B  
ffff

والمثال الآتى يضع الأمثلة السابقة فى برنامج واحد.:

مثال (3-6)

```
// Example 3-6.cs  
// Formatting Results  
  
using System;  
class Format  
{  
    static void Main()  
    {  
        string s;
```

```

// Currency:
s = "Currency";
Console.WriteLine("{0} Format:", s);
Console.WriteLine("{0:C}", 1.2);
Console.WriteLine("{0:C}", -1.2);

// Decimal:
s = "Decimal";
Console.WriteLine("\n{0} Format:", s);
Console.WriteLine("{0:D5}", 123);

// Floating -point:
s = "Floating -point";
Console.WriteLine("\n{0} Format:", s);
Console.WriteLine("{0:F2}", 12);
Console.WriteLine("{0:F0}", 12);
Console.WriteLine("{0:F0}", 12.3);
Console.WriteLine("{0:F2}", 12.3);

// General:
s = "General";
Console.WriteLine("\n{0} Format:", s);
Console.WriteLine("{0:G}", 1.2);
Console.WriteLine("{0}", 1.23);

// Numeric:
s = "Numeric";
Console.WriteLine("\n{0} Format:", s);
Console.WriteLine("{0:N}", 1230000000);

// Scientific:
s = "Scientific";
Console.WriteLine("\n{0} Format:", s);
Console.WriteLine("{0:E}", 12300000);

// Hexadecimal:
s = "Hexadecimal";
Console.WriteLine("\n{0} Format:", s);
Console.WriteLine("{0:X}", 123);
Console.WriteLine("{0:X}", 65535);
}
}

```

تنفيذ البرنامج:

```
Currency Format:
$1.20
($1.20)

Decimal Format:
00123

Floating-point Format:
12.00
12
12
12.30

General Format:
1.2
1.23

Numeric Format:
1,230,000,000.00

Scientific Format:
1.230000E+007

Hexadecimal Format:
7B
FFFF
```

**الحرفيات (string)**

من كانت له خبرة بلغة سي فإنه بلاشك يعرف "مشكلة" الحرفيات في اللغة ، فاللغة لا تحتوى على كلمة string كسائر اللغات التي سبقتها. والمشكلة تأتي من ضرورة إعلان الحرفي كمؤشر أو كمصفوفة من اللبئات وهذا يؤدي إلى اختلاط مفاهيم المؤشرات والحرفيات ، علاوة

على ما يتبع ذلك من مشكلات التعامل مع المؤشرات ، والأخطاء الناجمة عنها ، أو مشكلات نزع الذاكرة (Memory leak) التي يتعذر حلها في بعض الأحيان. إن سي# قد حلت هذه المشكلة بصورة قاطعة. فالحرفي ينتمي إلى أنماط المرجع ولكنك لا تحتاج أن تعرف عنه أكثر من ذلك ، فأنت تتعامل معه كما تتعامل مع سائر البيانات والمتغيرات ، بينما يتولى المترجم العمليات المعقدة التي تتم في الخلفية ، وأهمها جمع القمامة التي تتخلف عن استخدام المؤشرات.

### تعبيرات الحرفيات

لإعلان متغير حرفي استخدم النمط string كالإعلان التالي:

```
string myString;
```

ويمكنك شحن المتغير الحرفي أثناء الإعلان كالمثال الآتي:

```
string myString = "Welcome to the string world!";
```

ويأتي البيان الحرفي بداخل علامتي الاقتباس المزدوجة. ومن الجائز أن تضع بداخل علامتي الاقتباس أية لبنات بما في ذلك لبنات الهروب (

Escape characters) مثل لبنة السطر الجديد:

```
string myString = "My friend, \nWelcome to the string world!";
```

فهذا الحرفي يحتوي على العبارة الآتية:

```
My friend,
```

```
Welcome to the string world!
```

ولكي تضمن الحرفي أية علامات خاصة مثل علامة الشرطة المائلة أو

علامة الاقتباس نفسها فإنك تسبقها بشرطة مائلة كالمثال الآتي:

```
string myString = "To go to the next line use the new line
character: \"\\n\"";
```

إن هذا الحرفي يحتوى على العبارة التالية:

To go to the next line use the new line character: " \n"

وكما فى لغة سى ، فإن الشرطة المائلة المزوجة تستخدم للتعبير عن أسماء الفهارس (Derectories) مثل:

```
string myDir = "C: \\Documents\\Letterse\\friends.doc";
```

وفيما يلى ملخص بأهم لبنات الهروب المستخدمة فى لغى سى #.

جدول (3-7) لبنات الهروب (Escape Characters)

المعنى	لبنة الهروب
علامة اقتباس مفردة	'
علامة اقتباس مزدوجة	"
الشرطة المائلة	\\
المسح إلى الخلف (Backspace)	\b
الصفحة التالية	\f
السطر التالى	\n
من أول السطر	\r
بياضة (أفقية)	\t
بياضة رأسية	\v

### مؤثرات الحرفيات

يتطلب التعامل مع الحرفيات استخدام بعض المؤثرات نلخصها فيما يلى.

## مؤثرات الوصل + و +=

لوصل حرفيين استخدم المؤثر "+" كالمثال الآتي:

```
string s = "Hello" + " " + "World!";
```

بهذا يحتوى المتغير s على العبارة Hello World! وبالمثل يمكنك وصل المتغيرات الحرفية بنفس الطريقة:

```
string s = "Hello";
```

```
string space = " ";
```

```
string t = "World!";
```

```
string myString = s + space + t; // وصل الحرفيات بالترتيب
```

أو

```
s += space + t; // استخدام المؤثر المركب
```

وفى العبارة الثانية استخدمنا المؤثر المركب "+=" لإجراء عملية الوصل ووضع النتيجة فى نفس المتغير. أى أن العبارة الثانية تقول "اجمع المتغير s على المتغيرين space و t ثم ضع الناتج فى المتغير s". ومن البديهي أن وصل الحرفيات يتم تبعاً للترتيب المحدد فى العبارة. ولو أنك كتبت الطرف الأيمن بالصورة "t + space" فإن المسافة الخالية سوف تصبح فى ذيل العبارة.

## مؤثر التساوى ==

يستخدم المؤثر "==" لاختبار التساوى كالاتى:

```
Console.WriteLine(s == myString);
```

إن نتيجة العبارة السابقة هي القيمة True ، ومعناها أن محتويات المتغيرين s و myString متماثلة. والعكس هو القيمة False.

### المؤثر []

أما المؤثر [] فهو يؤدي إلى التوصل إلى اللبنة المفردة للحرفي (وهذا لا يختلف عن مفهوم مصفوفة الحرفيات في لغة سي). وعلى سبيل المثال يمكنك طباعة الحرف W من العبارة Hello World! على أساس انه اللبنة رقم 6 من العبارة (لاحظ أن العد يبدأ من الصفر):

```
Console.WriteLine(myString[6]);
```

أو

```
Console.WriteLine("Hello World!"[6]);
```

### @ المؤثر

يغنيك هذا المؤثر عن استخدام لبنات الهروب بداخل علامتي الاقتباس ، فعلى سبيل المثال ، بدلاً من كتابة التعبير الآتي:

```
string myDoc = "C: \\Documents\\Letters\\friends.doc";
```

يمكنك أن تكتب نفس المضمون بالطريقة المعتادة مع وضع المؤثر @ قبل الحرفي ، أي:

```
string myDoc = @"C: \Documents\Letters\friends.doc";
```

واستخدام هذا المؤثر قبل التعبير الحرفي ينسخ ما بين علامتي الاقتباس بالضبط بما في ذلك الانتقال إلى سطر جديد. وعلى سبيل المثال:

```
string a = @"Dear Sir,
```

```
I have read you manuscript 'Learn Pascal in Three Days', and I  
would like to inform you that we are interested in publishing the  
book.
```

Yours,  
M. M. Abdullah";

لو أنك طبعت محتويات هذا الحرفي فإنك تحصل على الآتي:

Dear Sir,

I have read you manuscript 'Learn Pascal in Three Days', and I would like to inform you that we are interested in publishing the book.

Yours,  
M. M. Abdullah

ولطباعة علامة الاقتباس المزدوجة بداخل الحرفي المسبوق بالموثر @  
اكتب علامتي اقتباس بدلاً من علامة واحدة مثل:

@ "He said, "" You should stop by when you can, "" Ok?"

عند طباعة محتويات هذا الحرفي تحصل على الآتي:

He said, " You should stop by when you can," Ok?

أما الاستخدام الأخير للعلامة @ فهو إضافتها كبداية إلى الكلمات المفتاحية مثل int و bool إلى آخره في حالة إذا ما أردت استخدامها كأسماء للمتغيرات ، أي @bool و @int.

مثال (3-7)

```
// Example 3-7.cs
// Strings

using System;

class myClass
{
    static void Main()
    {
```

```
bool isEqual;
string a = "\u0048ello my friend.\n";
string b = @"You can compose Unicode numbers using Escape
characters.
However, you cannot use the @ operator with that.";

// Print both a and b:
Console.WriteLine(a + b);

// Check for equality:
isEqual = (a == b);
Console.WriteLine("BTW, the equality is: {0}.", isEqual );
}
}
```

### تنفيذ البرنامج:

```
Hello my friend.
You can compose Unicode numbers using Escape characters.
However, you cannot use the @ operator with that.
BTW, the equality is: False.
```

### تدريب (3-5)

أكتب برنامجاً يطبع الآتى على الشاشة باستخدام الكود "يونيكود"  
للحروف الأبجدية A, B, C.  
A, B and C are the first three letters.

### قراءة المدخلات من لوحة الأزرار (Keyboard Input)

لقراءة البيانات من لوحة الأزرار استخدم أسلوب دوت.نت.  
Console.ReadLine وهو يقرأ كل ما هو مكتوب فى خط الأوامر عند  
الضغط على زر الإدخال ENTER. ولذلك نتوقع أن هذا الأسلوب يقرأ  
البيان فى صورة حرفى. ويستخدم الأسلوب ReaLine بالصورة الآتية:  
string myString = Console.ReadLine();

تؤدي هذه العبارة إلى قراءة البيان من لوحة الأزرار واختزانه في المتغير الحرفي `myString`. أنظر المثال التالي الذي يطبع رسالة تنبيه على الشاشة لتطلب منك أن تدخل عدداً ما ، ثم يستقبل البرنامج العدد ويختزنه في المتغير الحرفي `theNumber` ، ويطبعه على الشاشة باستخدام الرسالة المناسبة.

### مثال (3-8)

```
// Example 3-8.cs
// Reading from the keyboard

using System;

public class Keyboard
{
    public static void Main()
    {
        Console.WriteLine("Please enter a number: ");
        string theNumber = Console.ReadLine();
        Console.WriteLine("Your number is: {0}", theNumber);
    }
}
```

### مثال التنفيذ:

```
Please enter a number: 33
Your number is: 33
```

وقد يلزم تحويل الحرفي المدخل إلى أي نمط آخر من أنماط البيانات. وهذا يتم بأكثر من طريقة.

**تحويل الحرفي إلى عدد باستخدام الفصيلة Convert**

الطريقة الأولى هي استخدام فصيطة التحويل **System.Convert** من فصائل مكتبة دوت نت. وهو يستخدم كالاتى:

```
int myInt = Convert.ToInt32(myString);
```

وكما نرى أن هذه الصورة للأسلوب تحول الحرفى إلى النمط الصحيح Int32 وهناك صور أخرى للتحويل إلى الأنماط العددية الأخرى مثل:

```
Convert.ToInt64(myString); // طويل
```

```
Convert.ToDouble(myString); // مضاعف الدقة
```

```
Convert.ToSingle(myString); // حقيقى
```

```
Convert.ToDecimal(myString); // عشرى
```

كما يجوز أن تتم عملية التحويل مع إضافة بيان عددى إلى قيمة المتغير فى نفس الخطوة.

```
int myInt = Convert.ToInt32(theNumber) + 20;
```

وفى المثال التالى نستخدم تحويلات مختلفة إلى الأنماط العددية الحقيقية.

### مثال (3-9)

```
// Example 3-9.cs
// Converting strings to numbers

using System;

public class Keyboard
{
    public static void Main()
    {
        Console.Write("Please enter a number: ");
        string theNumber = Console.ReadLine();
        Console.WriteLine("Your string number is: {0}", theNumber );

        double d = Convert.ToDouble(theNumber);
```

```
float f = Convert.ToSingle(theNumber);
decimal c = Convert.ToDecimal(theNumber);

Console.WriteLine("Your decimal number is : {0}", c);
Console.WriteLine("Your double number is : {0}", d );
Console.WriteLine("Your float number is : {0}", f);
}
}
```

### تنفيذ البرنامج:

```
Please enter a number: 3.14
Your string number is: 3.14
Your decimal number is : 3.14
Your double number is : 3.14
Your float number is : 3.14
```

### تحويل الحرفي إلى عدد باستخدام الأسلوب Parse

أما الطريقة الثانية للتحويل فهي استخدام الأسلوب Parse من أساليب دوت.نت ، ويستخدم كالمثال الآتي:

```
int x = Int32.Parse(myString);
```

في هذه العبارة تم التحويل من حرفي إلى نمط صحيح Int32. وهناك تحويلات إلى الأنماط العددية الأخرى مثل:

```
Int64.Parse(myString) // الطويل
```

```
Single.Parse(myString) // الحقيقي
```

```
Decimal.Parse(myString) // العشري
```

```
Double.Parse(myString) // مضاعف الدقة
```

### مثال (3-10)

```
// Example 3-10.cs
// Parsing input
```

```
using System;

public class ParseClass
{
    static void Main()
    {
        Console.WriteLine("Please enter an integer number: ");
        string str = Console.ReadLine();

        long myLong = Int64.Parse(str);
        int myInt = Int32.Parse(str);
        double myDouble = Double.Parse(str);

        Console.WriteLine("Your long number is : {0}", myLong);
        Console.WriteLine("Your int number is: {0}", myInt);
        Console.WriteLine("Your double number is: {0}", myDouble);
    }
}
```

### تنفيذ البرنامج:

```
Please enter an integer number: 12
Your long number is: 12
Your int number is: 12
Your double number is: 12
```

ومن الجدير بالذكر أنك لا تستطيع إدخال عدد ذي علامة عشرية من لوحة الأزرار إلى هذا البرنامج ، لأن التحويل إلى النمط الصحيح أو الطويل في هذه الحالة يتعارض مع قواعد التحويل.

