الباب الثالث

أنماط البيانات

Data Types

محتويات الباب

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

أنماط لغة سى#

تنقسم الأنماط بصفة عامة فى لغة سى# إلى نوعين:

- **أنمـــاط القيمة (Value types)**: وهى أنماط المتغيرات التى تختزن قيم البيانات نفسها.
- **أنمـــاط المـــرجع (Reference types)**: وهى تمثل المتغيرات التى تخـــتزن عنوان الذاكرة الذى يشير إلى البيان الموجود فى حيز آخر من الذاكرة.

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

الأنماط المبنية فى اللغة

إن أنماط سى# المبنيّة فى اللغة ماهى إلاّ أسماء مستعارة لبعض الأنماط المعـــرفة فى المعمار دوت.نت ، وهى جميعاً موضحة بالجدول التالى حيـــث نـــرى اسم النمط وأمامه الاسم المؤهل للنمط المعرّف فى الحيز System بمكتبة فصائل دوت.نت.

جدول (1-3) الأنماط المبنية فى لغة سى#

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

* 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 سى

فـــى بـــرنامجك ، أى أن العبارتين التاليتين متساويتان تماماً فكلاهما تعلن عن متغير صحيح وتخصص له القيمة 25:

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

كذلـــك تتســـاوى كل من العبارتين التاليتين فى الإعلان عن متغير من النمط object:

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

كمـــا أن الأنمـــاط 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 الـــبوليانى (bool). كما تشمل الأنمـــاط المبتكـــــرة مثل المنشأ (struct) ، ومنشـــأ القائمة (Enumeration) وسوف نتعرف بهذه الأنماط تباعاً. وجميع أنماط القيمة موضحة بالملحق (أ).

وتختزن قيم أنماط القيمة فى حيز الذاكرة المسمى بالخزنة (Stack).

الخزنة (Stack):

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

وتنحدر جميع أنماط القيمة من فصيلة الهدف "object" وهو جذر شجرة الفصـــائل فـــى مكتـــبة فصائل دوت.نت. ومع ذلك فإن أنماط القيمة لا توّرث.

شحن المتغيرات

لا يمكـــنك اســـتخدام أى متغير محلى بدون شحنه بقيمة ما. والمقصود بالاستخدام هو دخول المتغير فى أى عملية مثل طباعة محتوياته. وتتم عملية شحن المتغيرات كالأمثلة 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 باستخدم 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();
```

وتتمـــيز 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 تخلـــق هدفـــا من المنشأ تستخدم الكلمة new التى تشحن أعضاء المنشأ

تلقائـــياً 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();
```

بهـــذه العـــبارة فإن جميع عناصر المنشأ (x و y فى هذه الحالة) قد تم شـــحنها بالقيمة 0 وهى القيمة سابقة التعريف. وسوف نقدم المزيد عن المنشئات فى الفقرات القادمة.

خصائص أنماط المرجع (Reference Types)

أمـــا نمـــط المرجع فهو يناظر المراجع (References) فى لغة سى++ حيـــث أن المتغـــير من نمط المرجع لا يحتوى على البيانات نفسها بل يحتوى على عنوانها. والمتغير المرجع يشغل حيزاً فى **الخزنة** (Stack) مـــثل متغيرات القيمة ، أما البيانات نفسها فهى توجد فى حيز آخر من الذاكرة وهو **الكوم** (Heap).

االكوم (Heap):

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

والآتى بعد أنماط المرجع المستخدمة فى سى#.

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

اسم النمط	الكلمة المستخدمة فى الإعلان
الفصيلة	class
الوصلة البينية	interface
المندوب	delegate
الهدف (الفصيلة الأساسية فى فصائل دوت.نت)	object
الحرفى	string

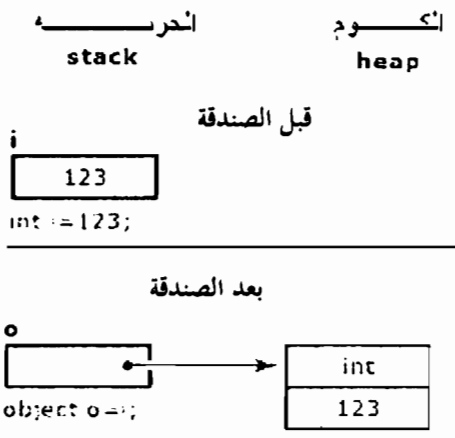
ومن الجائز التحويل ما بين أنماط القيمة وأنماط المرجع باستخدام عملية "الصندقة" (Boxing) والعملية العكسية "إلغاء الصندقة" (Unboxing).

الصندقة وإلغاء الصندقة (Boxing and Unboxing)

تتم عملية الصندقة بتحويل أى نمط قيمة إلى نمط الهدف (object) كالمثال التالى:

```
int i = 123;  // متغير صحيح
object o = i;  // تخزين المتغير فى متغير آخر من النمط object
```

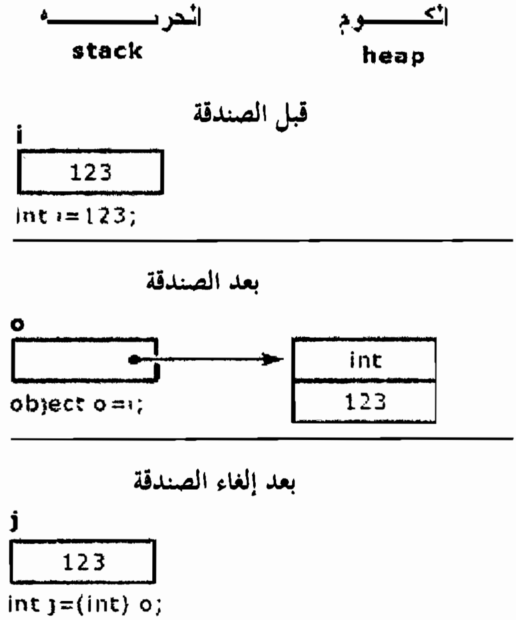
بموجب هذه العبارة يتم اختزان النمط int والقيمة 123 فى "الكوم" أما المتغير o فهو مرجع يشير إلى كل منهما. أنظر الشكل التالى:



لـــتحويل المتغـــير والبيان المصاحب له إلى نمط قيمة مرة أخرى فإننا نستخدم عملية إلغاء الصندقة ، وهى عبارة عن تحويل النمط object إلى النمط int (أو أى نمط آخر من أنماط القيمة) باستخدام الإسقاط كما فى العبارة التالية:

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

إن هذه العبارة تخلق متغيراً j من نمط القيمة فى "الخزنة" ويحتوى على نفس البيان 123. والشكل التالى 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 فى هذا المثال أننا غيرنا القيمة المختزنة فى المتغير المرجع أثناء عملية الصندقة وذلك بإضافة العدد 456 إلى القيمة الأصلية. ومع ذلك فهذا لم يغير من قيمة المتغير الأصلى i الذى ظل محتفظاً بقيمته الأصلية بعد الصندقة. أما المتغير j فهو يحتوى على نفس قيمة المرجع o.

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

الأنماط البسيطة (Simple Types)

يوضح الجدول التالى الأنماط البسيطة ، وهى تشمل الأنماط العددية (الصحيح "Integral" منها والحقيقى "Floating-point") علاوة على النمط

البولـــيانى "bool" ، ونمـــط اللبنة "char". ويطلق على هذه الأنماط اسم الأنمـــاط البســـيطة لتميـــيزها عن الأنماط المبتكرة التى يقوم البرنامج بتألـــيفها مثل المنشأ (struct) والقائمة (enum). ومن البديهى أن جميع الأنماط البسيطة أنماط قيمة.

جدول (3-3) الأنماط البسيطة

اسم النمط	السعة بالبايت	مدى البيان الذى يسمح به
bool	1	يحتوى إما على[1] true أو false
byte	1	من 0 إلى 255 (بدون إشارة)
char	2	من U+0000 إلى U+ffff (يونيكود Unicode)[2]
decimal	16	من $\pm 1.0 \times 10^{-28}$ إلى $\pm 7.9 \times 10^{28}$
double	8	من $\pm 5.0 \times 10^{-324}$ إلى $\pm 1.7 \times 10^{308}$
float	4	من $\pm 1.5 \times 10^{-45}$ إلى $\pm 3.4 \times 10^{38}$
int	4	من 2,147,483,648- إلى2,147,483,647
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

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

ملاحظات على الجدول:

(1) لا يوجد تحويل بين بيان من النمط bool والبيان العددى كما فى لغة سى++. أى أن القيمة false لا تساوى صفراً وكذلك القيمة true لا تساوى قيمة غير صفرية.

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

أمثلة على استخدام الأنماط البسيطة

فيما يلى بعض الإعلانات للأنماط البسيطة:

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

كما يمكنك الإعلان عن المتغير والتخصيص له فى نفس الوقت باستخدام عبارات مثل:

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

البيانات العددية (Numeric Data)

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

التعبيرات الحسابية

تسـتطيع فـى لغـة سى# أن تبنى ما تشاء من التعبيرات الحسابية باستخدام البيانات والمتغيرات والمؤثرات.

مؤثرات العمليات الحسابية الأساسية

تبـنى التعبيرات الحسابية باستخدام المؤثرات الحسابية مثل + ، − ، * ، / وهـى المؤثـرات الحسابية البسيطة المعروفه فى سائر اللغات لإجـراء الجمع والطرح والضرب والقسمة. وعلى سبيل المثال يمكنك كتابة تعبير حسابى مثل:

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

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

```
6 + 4 * 5;
```

فـــى هذا التعـ فإن المقدار 5*4 يتم تقييمه أولاً إلى 20 ، ثم يجمع الناتج على 6 أى ون النتيجة 26. ويمكنك أن تستخدم الأقواس لتغيير الأولوية. والأقوا 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) والنقـــصان (Decrement) المستخدمة فى لغة سى++ ، انظر المثال الآتى:

```
x ++;  // زيادة محتويات المتغير x بمقدار 1
x --;  // إنقاص محتويات المتغير x بمقدار 1
```

وتـــتوقف قـــيمة 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);  // 110   :النتيجة
    z =  --z +x;
    Console.WriteLine(z);  //  109 :النتيجة
  }
}
```

تنفيذ البرنامج:

```
110
109
```

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

تدريب (3-1)

ابدأ بالتخصيص للمتغيرات الاتية:

```
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 (y++) لتوضيح المعنى (للقارئ).

كيف تحصل على اسم النمط

للحصول على أنماط المتغيرات أو التعبيرات استخدم أسلوب دوت.نت **GetType()** بالطريقة الآتية:

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

- حيث myVariable: المتغير أو التعبير المراد طباعة نمطه.

والنتيجة التى نحصل عليها هى 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 هى النتائج التى تحصل عليها:

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

تقييم التعبيرات ذات الأنماط المختلطة

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

- إذا كـــان التعبير محتوياً على بيان من النمط double فإن النتيجة تكون من النمط double.
- إذا لـــم 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)

يلـــزم مـــع بعض الأنماط أن تضيف حرفاً ما إلى نهاية البيان العددى (لاحقة Suffix) لتمييزه عن بقية الأنماط.

الأنماط الحقيقية

يعـــامل المترجم الكسور على أنها من النمط المضاعف الدقة (double) مالم تميزها بلاحقة ما.

- مع النمط العشرى (decimal) نستخدم الحرف M كالآتى:

```
23.4M;
```

- مع النمط الحقيقي (float) نستخدم الحرف F كالآتى:

```
23.4F
```

ولـــو أنـــك كتبت العدد بدون أى حرف فإنه يعامل على أنه من النمط double.

وعـــند الإعـــلان عن أحد هذه الأنماط فإن استخدام اللاحقة ضرورى ، وإلاّ 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:

بالـــرغم مـــن أنـــه يمكنك استخدام الحروف الكبيرة أو الصغيرة لكتابة اللواحق مثل F ، M وغيرها ، لكننا سوف نستخدم الحروف الكبيرة فى هذا الكتاب.

مثال (3-4)

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

using System;

class MyPoint
{
  static void Main()
  {
    decimal myDc = 23.4M;
    float myFl = 2  3.4F;
    double myDb = 23.4;

    Console.Write("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 الحرفى المراد طباعته على بضعة أسطر؟ إنك تستطيع بدلاً من ذلك أن تستخدم علامة السطر الجديد (\n) بداخل علامتى الاقتباس. والعلامة \n هى نفسها المستخدمة فى لغة سى++.

تدريب (3-3)

اختبر 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 لاتحتاج معها إلى إضافة حرف إلى نهاية الثابت العددى 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 العددى. وبحسب قيمة العدد يتم تخزينه فى المخزن الماسب كالآتى:

- عند استخدام الحرف L: يختزن العدد على أنه من النمط long أو ulong بحسب قيمته. وعلى سبيل المثال فإن التعبير الآتى 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 هى النتائج التى تحصل عليها:

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

تحويل الأنماط (Conversion)

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

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

تتضـــمن هـــذه الإعلانات عملية تحويل من النمط الصحيح إلى الأنماط الحقيقـــية float و double. ولكن هذه العمليات تتم فى 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 ــ كما ذكرنا فى الفقرات السابقة) فى مخزن أقل من حجمه وهو مخزن الـــنمط float. ولذلـــك فـــإن المترجم يعترض على هذه العبارة ولا يتم الترجمة. (يمكنك استخدام اللاحقة F مع العدد كما ذكرنا من قبل ، وهى تؤدى إلى تحويل البيان العددى نفسه).

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

ولنعتـــبر الإعلانـــات التالـــية 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
```

والســـبب فـــى ذلك أنه لا يوجد تحويل ضمنى من النمط int أو double إلى النمط short فهى جميعاً أكبر من مخزن النمط short.

ولإتمام عملية التحويل فإنه لابد من استخدام الإسقاط ، أى:

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

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

ولتحزين بيان عددى صحيح فى متغير فلابد من التأكد من سعة المخزن ، وعلى سبيل المثال فإن العبارة التالية تحاول تخزين عدد صحيح (int) فى المخزن المخصص للنمط القصير (short) ولكن العدد المراد تخزينه أكبر من سعة المخزن:

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

ولا يجوز استخدام الإسقاط فى 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.

اللبنات (char)

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

- اللبنة الحرفية الصريحة مثل:

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

- أو رقم سداسى عشر (Hexadecimal) مثل:

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

لاحظ ان الرقم المكتوب بالكود السداسى عشر يأتى مسبوقاً بالعلامة \x.

- أو رقم يونيكود مثل:

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

لاحظ ان الرقم المكتوب باليونيكود يأتى مسبوقاً بالعلامة \u.

- أو بتحويل الكود آسكى باستخدام الإسقاط مثل:

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

وفى جميع الأحوال نستخدم علامتى الاقتباس المفردة لاحتواء اللبنة.

يوضـــح المثال التالى طريقتين لتخزين اللبنة 'A' باستخدام الكود العددى 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 هى 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 العددى فى خانتين عشريتين ، كما تتطلب بعض التطبيقات كتابة علامة النقود (الدولار أو الجنيه) بجانب النتيجة. وهناك طريقة لإجراء

الفورمـــات تستخدم مع أسلوب الطباعة Write أو WriteLine نقدمها فى هـــذه الفقـــرة. كما أن الملحق "و" يلخص عمليات الفورمات فى جدول يمكنك استخدامه كمرجع سريع.

فورمات العملة (Currency)

لطـــباعة العدد مصحوباً بعلامة العملة مع إضافة الأصفار اللازمة فى الخانات العشرية استخدم لبنة الفورمات C أو c كالمثال الآتى:

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

إن العدد 1.2 يظهر فى الخرج بالصورة 1.20$.

ولو كان البيان العددى سالبا (1.2-) كما فى المثال التالى:

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

فإنه سوف يظهر بين قوسين بالصورة (1.20$).

الفورمات العشرية (Decimal)

لطـــباعة العدد مسبوقاً بعدد معين من الأصفار استخدم لبنة الفورمات D أو d كالمثال الآتى:

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

ونتـــيجة هذه العبارة هى 00123. أى أن عدد الأرقام فى النتيجة يحددها الرقم الذى على يمين الحرف D.

فورمات العلامة العشرية الثابتة (Fixed-point)

لطـــباعة العدد محتوياً على عدد معين من الخانات العشرية استخدم لبنة الفورمات F أو f كالأمثلة 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)

لطـــباعة 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
```

وهى نفس النتائج التى نحصل عليها باستخدام فورمات على الإطلاق ، مثل:

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

الفورمات العددية (Numeric)

لطـــباعة العـــدد محـــتوياً على فواصل بين الأرقام وعلامتان عشريتان استخدم لبنة الفورمات N أو n كالمثال الآتي:

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

ونتيجة هذه العبارة هى العدد:

1,230,000,000.00

الفورمات العلمية (Scientific)

لطـــباعة العدد بالصورة الأسية المستخدمة مع الأرقام الكبيرة ولا سيما فى التطبيقات العلمية ، استخدم لبنة الفورمات E أو e كالمثال الآتى:

```
Console.WriteLine("{0:E}", 12300000);
```

ونتيجة هذه العبارة هى العدد:

1.230000E+007

الفورمات السداسى عشر (Hexadecimal)

لطـــباعة العـــدد بـــالكود السداسى عشر استخدم لبنة الفورمات X أو x كالأمثلة الآتية:

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

ونتيجة هذه العبارات بالترتيب هى:

```
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.WriteLi ne("\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);
  Conso le.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 سى فإنه بلاشك يعرف "مشكلة" الحرفيات فى اللغـــة ، فاللغة لا تحتوى على كلمة string كسائر اللغات التى سبقتها. والمشـــكلة تـــأتى من ضرورة إعلان الحرفى كمؤشر أو كمصفوفة من اللبـــنات وهذا يؤدى إلى اختلاط مفاهيم المؤشرات والحرفيات ، علاوة

علـــى مـــا يتـــبع ذلك من مشكلات التعامل مع المؤشرات ، والأخطاء الـــناجمة عنها ، أو مشكلات نشع الذاكرة (Memory leak) التى يتعذر حلها فى بعض الأحيان. إن سى# قد حلت هذه المشكلة بصورة قاطعة. فالحرفى ينتمى 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 التى تتم فى الخلفية ، وأهمها جمع القمامة التى تتخلف عن استخدام المؤشرات.

تعبيرات الحرفيات

لإعلان متغير حرفى استخدم النمط string كالإعلان التالى:

```
string myString;
```

ويمكنك شحن المتغير الحرفى أثناء الإعلان كالمثال الآتى:

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

ويأتى البيان الحرفى بداخل علامتى الاقتباس المزدوجة. ومن الجائز أن تضـــع بداخل علامتى الاقتباس أية لبنات بما فى ذلك لبنات الهروب (Escape characters) مثل لبنة السطر الجديد:

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

فهذا الحرفى يحتوى على العبارة الآتية:

```
My friend,
Welcome to the string world!
```

ولكى تضمن الحرفى أية علامات خاصة مثل علامة الشرطة المائلة أو علامة الاقتباس نفسها فإنك تسبقها بشرطة مائلة كالمثال الآتى:

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

إن هذا الحرفى يحتوى على العبارة التالية:

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

وكمـــا فى لغة سى ، فإن الشرطة المائلة المزدوجة 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 (7-3) لبنات الهروب (Escape Characters)

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

مؤثرات الحرفيات

يتطلـــب الـــتعامل مع الحرفيات استخدام بعض المؤثرات نلخصها فيما يلى.

مؤثرات الوصل + و =+

لوصل حرفيين استخدم المؤثر "+" كالمثال الآتى:

```
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 البديهى 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 كالآتى:

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

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

المؤثر []

أما المؤثر [] فهو يؤدى إلى التوصل إلى اللبنات المفردة للحرفى (وهذا لا يختلف عن مفهوم مصفوفة الحرفيات فى لغة سى). 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 علامتى الاقتباس ، فعلى سبيل المثال ، بدلاً من كتابة التعبير الآتى:

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

يمكنك أن تكتب نفس المضمون بالطريقة المعتادة مع وضع المؤثر @ قبل الحرفى ، أى:

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

واستخدام هذا المؤثر قبل التعبير الحرفى ينسخ ما بين علامتي الاقتباس بالضبط بما فى 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 الحرفى فإنك تحصل على الآتى:

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 الحرفى المسبوق بالمؤثر @ اكتب علامتي اقتباس بدلاً من علامة واحدة مثل:

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

عند طباعة محتويات هذا الحرفى تحصل على الآتى:

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

أمـــا الاســـتخدام الأخـــير للعلامة @ فهو إضافتها كبادئة إلى الكلمات المفتاحـــية مـــثل int و bool إلى آخره فى حالة إذا ما أردت استخدامها كأسماء للمتغيرات ، أى @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 برنامجا 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();
```

تـــؤدى هـــذه العـــبارة إلى قراءة البيان من لوحة الأزرار واختزانه فى المتغـــير الحرفى myString. أنظر المثال التالى الذى يطبع رسالة تنبيه علـــى الشاشة لتطلب منك أن تدخل عدداً ما ، ثم يستقبل البرنامج العدد ويخـــتزنه فـــى المتغـــير الحرفى 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 Mai n()
  {
    Console.Write("Please enter a number: ");
    string theNumber = Console.ReadLine();
    Console.WriteLine("Your number is: {0}", theNumber );
  }
}
```

مثال للتنفيذ:

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

وقـــد يلزم تحويل الحرفى المدخل إلى أى نمط آخر من أنماط البيانات. وهذا يتم بأكثر من طريقة.

تحويل الحرفى إلى عدد باستخدام الفصيلة Convert

الطـــريقة الأولى هى استخدام فصيلة التحويل **System.Convert** من فصائل مكتبة دوت نت. وهو يستخدم كالآتى:

```
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 الحرفى إلى عدد باستخدام الأسلوب Parse

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

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

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

```
Int64.Parse(myString)  // الطويل
Single.Parse(myString)  // الحقيقى
Decimal.Parse(myString)  // العشرى
Double.Parse(myString)  // مضاعف الدقة
```

مثال (3-10)

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

```
using System;

public class ParseClass
{
  static void Main()
  {
     Console.Write("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 ذى علامة عشرية من لوحـــة الأزرار إلى هذا البرنامج ، لأن التحويل إلى النمط الصحيح أو الطويل فى هذه الحالة يتعارض مع قواعد التحويل.