

This file has been cleaned of potential threats.

To view the reconstructed contents, please SCROLL DOWN to next page.

مقدمة للبرمجة

بالسي شارب

المحتوى :

٧	١- عن الكتاب
٨	٢- مقدمة
١١	٣- إطار العمل
١٦	٤- بنية البرنامج في السي شارب
٢١	٧- الإدخال والإخراج
٢٦	٨- أنواع البيانات
٢٩	٩- المتغيرات
٣٩	١٠- تحويل انواع البيانات
٤٤	١١- التعليقات
٤٦	١٢- المعاملات الرياضية والمنطقية
٦١	١٣- الجُمْل الشرطية (switch - if/else....else if)
٧٧	١٥- حلقات التكرار (For / foreach / While / do....while)
١٠٠	١٧- المصفوفات (Arrays)
١٢٧	١٨- النصوص (Strings)
١٣٥	١٩- مقدمة في البرمجة كائنية التوجه (OOP)
١٣٧	٢٠- التغليف (Encapsulation)
١٤٣	٢١- static members

١٤٦	٢٢- الدوال (Methods or Functions)
١٧٤	٢٣- الدالة ذاتية الإستدعاء (Recursion function)
١٨٠	٢٤- (object , var , dynamic)
١٨٧	٢٥- Classes
٢٠١	٢٦- الخصائص (set , get)

عن الكتاب

هذا الكتاب مُعدّ للمبتدئين لكي يساعدهم على فهم أساسيات لغة السي شارب بطريقة مبسّطة وسهلة، ولا يشترط أن يكون القارئ له أي معرفة مسبقة بالبرمجة، وأهم خطوة بجانب القراءة هي التطبيق العملي المستمر لتحصل على أكبر قدر من الفهم والإستيعاب والتحصيل.

عن اللغة

لغة السي شارب لغة بسيطة جداً وسهلة وحديثة ولها أغراض كثيرة، فهي طُوّرت بواسطة شركة مايكروسوفت (Microsoft) بقيادة أندريس هيلسبرج، اللغة تُمكنك من عمل تطبيقات للويب والهاتف وسطح المكتب، وتعمل شركة مايكروسوفت على تطويرها دائماً، وهي واحدة من أشهر لغات البرمجة وأكثرهم إستخداماً. بدأوا في تطوير اللغة عام ١٩٩٩ ، وأطلقوا عليها لغة كool أي :

COOL ----> C Object Oriented Language

ثم غيروا الاسم لسي شارب وأعلنوا عنها عام ٢٠٠٠.

ولغة سي شارب (C#) هي تطوير وإمتداد للغة ++C و C، وهي تعمل بالبرمجة كائنية التوجه بشكل كلي.

1

مقدمة

إن سي شارب إحدى لغات البرمجة المصممة للعمل على البنية التحتية المشتركة للغات البرمجة تسمى

(Common Language Infrastructure) ← (CLI)

معنى هذا المصطلح أنها تُتيح للمبرمج إمكانية استخدامها لإنتاج مكتبات تتوافق مع المواصفات والخصائص الشائعة للبنية التحتية أي انه يحتوي على الكود التنفيذي الذي يتيح له استخدام شتى لغات البرمجة عالية المستوى علي منصات عمل مختلفة.

التسلسل الهرمي للبيانات

Bits- هي اصغر وحدة قياس في الكمبيوتر وهي تأخذ قيمتين فقط إما 0 أو 1.

تُعبّر الأصفار والأحاد عن إشارات كهربائية، وهي ما يعمل بها الكمبيوتر.



Characters - وهي تتضمن الحروف وتساوي ٢ بايت حيث البايت يساوي ٨ بت

ثم تتحول الحروف إلى أرقام وأحاد (لغة الآلة) لكي يفهمها الكمبيوتر ويتم ذلك بـ :

ASCII → (American Standard Code for Information Interchange)

فكل حرف من (A إلى Z) وكل رقم من (0 إلى 9) وكل رمز في الكمبيوتر مثل & ^ \$ # @ له عدد عشري يناظره ثم يتحول إلى أرقام وأحاد في النهاية لكي يفهمه الكمبيوتر .

ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	*	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

١- بت = (٠ | ١)

٢- بايت = ٨ بت

٣- كيلو بايت = ١٠٢٤ بايت

٤- ميغا بايت = ١٠٢٤ كيلو بايت

٥- جيجا بايت = ١٠٢٤ ميغا بايت

٦- تيرا بايت = ١٠٢٤ جيجا بايت

٧- بيتا بايت = ١٠٢٤ تيرا بايت

٨- إكسا بايت = ١٠٢٤ بيتا بايت

٩- زيتا بايت = ١٠٢٤ إكسا بايت

١٠- يوتا بايت = ١٠٢٤ زيتا بايت

Fields (حقول) - تتكون من حروف او أرقام فمثلا تُعبر عن اسم الشخص أو عمره.

Records (سجل) - يحتوي على مجموعة من الحقول معاً.

Files (ملفات) - تتكون من مجموعة من السجلات.

Database (قواعد البيانات) - عبارة عن مجموعة من البيانات وقد تكون بيانات ضخمة لكن يسهل الوصول إليها والتعامل معها بسرعة وتتكون من حقول وسجلات وهذه البيانات تُخزن في جداول بسيطة لها اسم وخصائص.

Big Data (بيانات ضخمة) - كم هائل من البيانات التي تُنتج في العالم يومياً.

2

إطار العمل

(The .Net Framework)

الـ (دوت نت) .Net هي إطار عمل أو منصة لتنفيذ التطبيقات وتحتوي على مكتبات توفر العديد من الامكانيات التي تستخدم لبناء تطبيقات كبيرة بالسي شارب سواء تطبيقات للويب أو لسطح المكتب أو الهاتف سواء (windows ، ios ، android ، phone) وذلك من خلال xamarin وهي عبارة عن منصة لتطوير تطبيقات الهواتف المحمولة حيث يعمل التطبيق الذي أنشئ بواسطتها على كل أنظمة تشغيل الهواتف، الـ .Net صُممت لتكون منصة عمل لمختلف التطبيقات، تستطيع إستخدامها في بناء تطبيقات من خلال اللغات الآتية :

C# , C++ , Visual Basic , Python and JavaScript

وهي تحتوي على كل المكتبات المبنية في هذه اللغات وبها كل الموارد التي يمكنك من إنشاء تطبيق من خلالها.

محتويات إطار عمل (.Net) :

CLR (Common Language Runtime) - هي عبارة عن آلة افتراضية مسؤولة عن إدارة الكود مع الذاكرة المؤقتة (RAM) والمعالج (processor) تُحول الكود إلى لغة الآلة، بعدها هذا الكود يستطيع العمل على أي لغة برمجة اخرى، وهو يدير الذاكرة أيضا ويعالج بها الاخطاء ويحتوي على " جامع النفايات ".

GC (Garbage Collector) - يجمع البيانات المهملة والاماكن الغير مستخدمة في الذاكرة ويقوم بحذفها وتنقية الذاكرة فيجعلها تعمل بكفاءة أكثر.

رحلة الكود البرمجي داخل الـ .Net :

أولاً يُعالج الكود فيتحول إلى لغة وسيطة

(MSIL → Microsoft Intermediate Language)

لأن السي شارب من اللغات عالية المستوى فتحتاج إلى لغة وسيطة قبل أن تتحول إلى لغة الآلة.

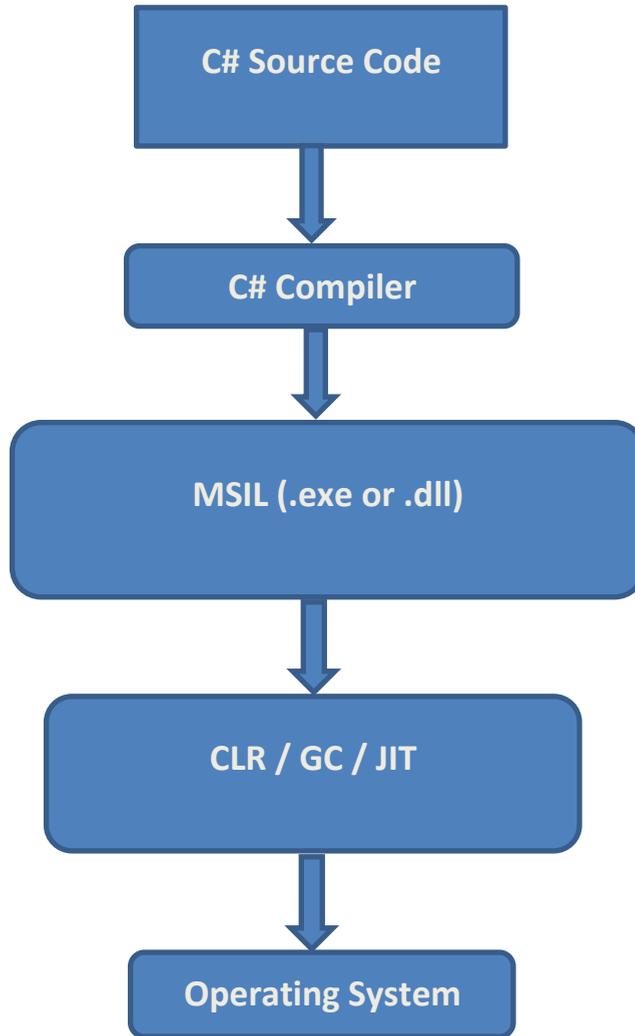
ملاحظة : مكونات تطبيقات اللغة الوسيطة وُضعت في البرنامج التنفيذي (الملف الذي يُنفذه الكمبيوتر في نهاية المهام).

ثانياً عندما يُنفذ البرنامج، يمر على معالج اخر يُعرف بـ just-in-time (JIT) وهذا أيضاً يكون بداخل CLR

ويعمل على ترجمة اللغة الوسيطة في الملف التنفيذي إلى لغة الآلة (0 , 1).

ملاحظة :

أي أمر يأخذه الكمبيوتر يتحول إلى اللغة التي يفهمها الكمبيوتر ويستطيع التعامل معها وهو لا يفهم غيرها وهي (0 , 1) أي شيء يتحول إلى الأصفار والآحاد.



وتوجد أيضا أداة داخل إطار عمل الـ (.Net) تسمى :

CTS (Common Type System)

وهي جزء من الـ CLR وهي عبارة عن أداة تستطيع تشغيل أكثر من لغة برمجة معاً بمكتبات مختلفة.

وتحتفظ بالاساسيات بداخلها وعندما تعمل أى لغة تعود هذه الأداة لتلك الاساسيات الخاصة بهذه اللغة لكي تعمل على إطار العمل حيث أن كل لغة برمجة يكون لها قواعد خاصة بها في الكتابة ويكون لكل لغة العديد من المكتبات التي تحتوي على تعريف لكل شئ خاص باللغة.

<https://msdn.microsoft.com/ar-sa/library/>

إن لم تستطع فهم هذه المعلومات فلا بأس، سنفهمها فيما بعد، فمحتوى الكتاب غير معتمد عليها، لكني تعمّدت قولها لكي تعرف كيف يعمل البرنامج الذي تكتب عليه الكود وماذا يحدث مع الكود الذي تكتبه وما يحدث وراء الشاشة وكيف يفهمه الكمبيوتر ويظهر على شكل برنامج له واجهه ويستطيع المستخدم العادي التعامل معه، سنبدأ في بناء هيكل أول برنامج في لغة السي شارب .

إذاً أين يُكْتَب الكود؟

يمكننا كتابة الكود على أي محرر نصوص ويكون إمتداد الملف الذي نكتب به الكود (.cs)

أو على بيئة تطوير متكاملة (IDE (Integrated Development Environment)

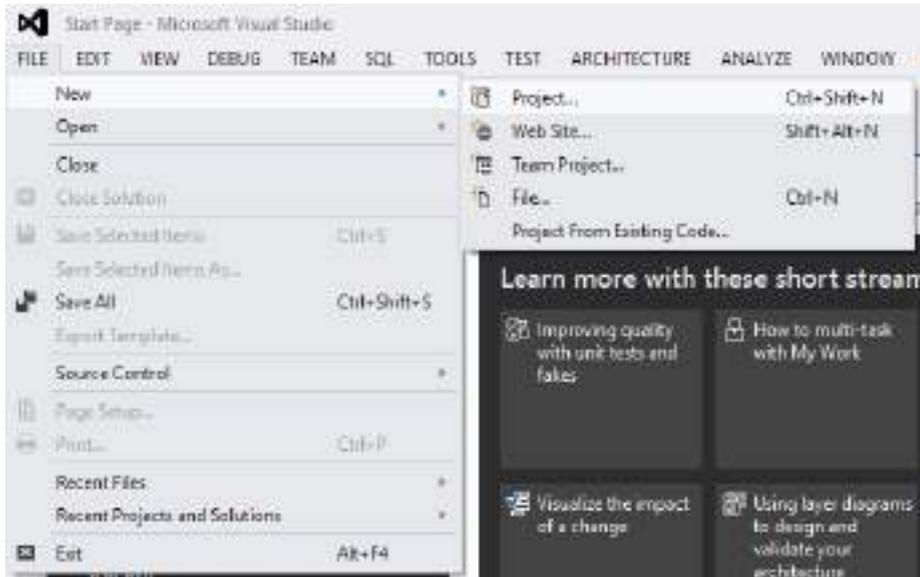
وهذه البيئة (visual studio) هي المعتمدة من شركة مايكروسوفت في التطوير.

يمكنك أن تُحمِلَ أي إصدار منه.

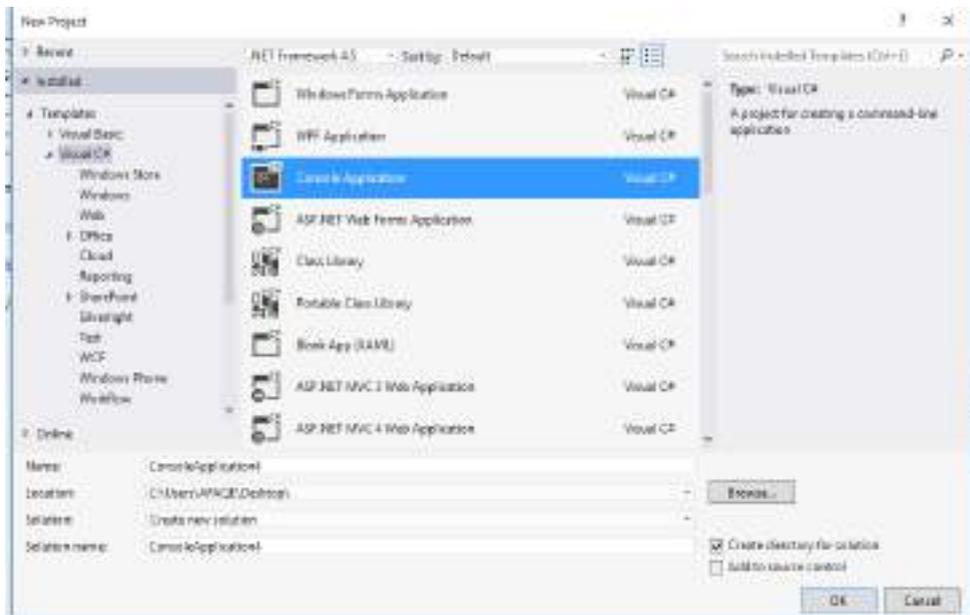
بعد تحميل البرنامج وتثبيته على جهازك ...

اولاً نفتح برنامج visual studio من قائمة Start

ثم نضغط على File بعدها ندخل إلى New ثم نختار project



بعدها نختار لغة C# ثم Console Application



سيفتح أمامك

```

ConsoleApplication3\Program
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ConsoleApplication3
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            // تكتب الكود هنا
14        }
15    }
16
17
18
19 }
20

```

using System;

namespace + اسم البرنامج

First_project إذا اسميت البرنامج

سيكون كالتالي:

namespace First_project

{

 Class Program

 {

 static void Main(string[] args)

 {

 Console.WriteLine("Hello World");

 }

 }

}

بعد الإنتهاء من كتابة الكود سنضغط على كلمة Start لكي نقوم بمعالجة الكود و تنفيذه.

أو يمكننا الضغط على (ctrl+f5)

ملاحظة : بعض إصدارات برنامج visual studio تفتح الشاشة السوداء (Console) وتغلق بسرعة قبل رؤية نتيجة الكود،

يمكنك أن تثبت الشاشة عن الإغلاق بكتابة (Console.ReadKey() ;)

أو (Console.ReadLine() ;)

وتكتب هذه الجملة في آخر سطر في الكود في دالة الـ Main() ...

```
namespace First_project
{
    Class Program
    {
        static void Main(string[] agrs)
        {
            Console.WriteLine("Hello World");
            Console.ReadKey( );
        }
    }
}
```

لكي نفهم هذا البرنامج يجب أن نحلله أولاً.

ما معني هذه الكلمات...؟

(using , namespace , class , Main)

اتفقنا في البداية على أن لغة السي شارب (C#) إنما هي مبنية على اللغتين C++ و C وتعمل بشكل كلي بالبرمجة كائنية التوجه.

إذاً ما معنى البرمجة كائنية التوجه؟ وما هو إختلاف السي شارب عن السي؟ وما هي تقنيات البرمجة؟

لنتعرف على تقنيات البرمجة أولاً :

١- البرمجة الخطية (Linear Programming)

وهي كانت عبارة عن أن الكود يُكتب في صفحة واحدة وبعض الاجزاء في الكود قابلة للتكرار بدون شئ يجعلنا نستغني عن هذا التكرار في جزء معين من الكود، فمثلاً أنك تكتب بعض الاسطر من الكود لتنفيذ عملية معينة ثم تُكمل الكود وبعدها تقابلك مشكلة تحتاج إلى نفس العملية السابقة لكن في جزء مختلف من البرنامج، حينها لا تستطيع إلا أن تعيد كتابة هذا الجزء مرة أخرى، وأيضاً عملية البحث عن الاخطاء كانت صعبة جداً وتأخذ الكثير من الوقت، كود يحتوي على آلاف الاسطر فلم يكن من السهل التعامل مع الكود في عمليات التعديل وكان الكود غير مرتب، وُجِدَت فكرة البرمجة الخطية لحل المشاكل المتعلقة بالرياضيات ولتسهيل حلها.

٢- البرمجة الهيكلية (Structure Programming)

ظهر بعض التطوير في هذه الوقت واخترعوا الدوال أو الوظائف (Functions) وهي تُمكنك من كتابة (عملية معينة) عدد من الأسطر مرة واحدة وتستدعيهم في كل مره احتجت إليهم دون الحاجة إلى إعادة كتابة هذه الاسطر (العملية)، فقط نكتب اسم الدالة، لكن أيضاً وُجِدَ بعض التقصير فيها، حيث يصعب فصل البيانات عن العمليات، ولا توجد حماية للبيانات من حدوث الأخطاء، والتعديل بها ينتج عنه أخطاء في أجزاء أخرى من الكود، ومن اللغات التي تعمل بالبرمجة الهيكلية لغة C.

٣- البرمجة كائنية التوجه (Object Oriented Programming)

وأخيراً ظهرت تقنية جديدة في كتابة الكود تُمكنك من تنظيم الكود وأختصرت جداً في حجم الكود وأصبح مرتباً وسهلاً عمليات البحث عن الأخطاء وساعد المطورين على العمل بأريحية ووفرت الكثير من الوقت والجهد أيضاً، وأصبح كل شيء منفصلاً عن الآخر حيث تتم عملية التعديل والتغيير في الكود بدون حدوث أخطاء مترتبة على هذا التغيير، وأضيفت الكثير من التقنيات التي تحافظ على البيانات وتغليفها وإخفاء البيانات والتحكم الكامل بها، ومن اللغات التي تعمل بالبرمجة كائنية التوجه (البرمجة الشيئية) لغة C++ و C# و Java و Python ومعظم لغات البرمجة الحديثة.

Namespace

لنتخيل أننا كتبنا مجموعة من الأسطر البرمجية في دالة (Function)، وأنشأنا أكثر من دالة، إذاً نحتاج إلى شيء يدير هذه الدوال معاً، اسمه class.

لا تشغل تفكيرك بمعنى كلمة class، سنفهمه فيما بعد، كل ما أريدك أن تعرفه عن الـ class أنه يمكن أن يحتوي على أكثر من دالة ويمكن إنشاء أكثر من class في نفس المشروع، إذاً نحتاج إلى شيء ليجمع كل هذه الـ classes مع بعضها، سنضعهم بداخل Namespace فيكون الكود مرتب أكثر.

إذاً فالـ (namespace) هي عبارة عن حاوية لمجموعة من الـ classes، ويمكن أن يحتوي المشروع على أكثر من namespace.

ولنستطيع رؤية وإستخدام الـ classes التي بداخل الـ namespace

يجب أن نستدعي الـ namespace بإستخدام كلمة using .

Main

تسمى الدالة الأساسية التي يبدأ منها البرنامج في تنفيذ الكود، فمثلاً إذا كتبت مئات العمليات والإجراءات خارج الدالة الأساسية فلن يتم تنفيذهم أولاً بل يبدأ البرنامج في تنفيذ ما بداخل الدالة الأساسية.

```
static void Main(string[ ] agrs)
{
    Console.WriteLine(" C# " );
}
```

سنفهم معنى كلمة `string[] args` في الفصل الخاص بالدوال، يُمكن الإستغناء عنها في دالة الـ `Main`، نستطيع كتابة أقواس الدالة بدون كتابتها.

تكون بالشكل التالي :

```
static void Main( )
{
    Console.WriteLine(" C# " );
}
```

3

الإدخال والإخراج

(Input and Output)

عمليات الإدخال والإخراج هي التي تُمكن المستخدم من التعامل مع البرنامج أو التطبيق، على سبيل المثال إذا كنت من مستخدمي موقع الفيس بوك فإنك أولاً تُدخل البريد الإلكتروني وكلمة المرور فهذه تُعد عملية إدخال لك كمستخدم للموقع، والبيانات التي تظهر فيما بعد فهذه عملية إخراج.

```
using System ;
namespace First_project
{
    Class Program
    {
        static void Main(string[ ] agrs)
        {
            Console.ReadLine( );
        }
    }
}
```

كلمة Console عبارة عن class بداخل Namespace (System) `ReadLine()` - عبارة عن دالة تُستخدم لعمليات الإدخال وتستقبل نصوص فقط.

ويجب ان نضع فاصلة منقوطة في نهاية كل جملة في البرنامج (;) وإلا سيحدث خطأ.

أنواع الأخطاء في الكود :

١- خطأ في طريقة كتابة الكود (Syntax error) وذلك يكون خطأ في قواعد كتابة الكود.

٢- خطأ منطقي (Logic error) وفي هذه الحالة يكون الكود مكتوب بطريقة صحيحة كقاعدة ولكنه غير منطقي مثل قسمة واحد على صفر.

٣- خطأ أثناء التشغيل (Runtime error) هذا الخطأ يحدث أثناء تشغيل البرنامج، مثل وجود فائض في البيانات المُدخلة.

نسيان الفاصلة المنقوطة يُعد خطأ في طريقة الكتابة أي (Syntax error).

```
using System ;
namespace First_project
{
    Class Program
    {
        static void Main(string[ ] agrs)
        {
            Console.WriteLine(" C# ");
        }
    }
}
```

() WriteLine - هي دالة لطباعة المخرجات

Line - تُكتب لكي يُنهي السطر الحالي ويبدأ في سطر جديد.

ويمكن أن نكتب فقط Write() لكن الأسطر التي تليها في عملية الطباعة ستكون متلاصقة بها.

ويوجد طريقتان للطباعة :

١ - (Placeholder)

وتكون كالتالي :

```
Console.WriteLine(" Hello World {0} " , 5 , 20 ) ;
```

وفي هذه الحالة سيطلع Hello World + أول مُدخَل للدالة بعد الفاصلة

وهي القيمة (5)

وإذا أردنا طباعة ثاني مُدخَل سنكتب {1} وحينها لن يعرض أول مُدخَل وسيعرض القيمة 20

حيث أن أول ترقيم (index) يبدأ بالقيمة صفر { 0 }.

ويُمكنك أن تطبع أي شيء بالشكل الذي تريده فمثلاً :

```
Console.WriteLine(" My Name is {0} and my age is {1}" , " Ali " , 50 ) ;
```

في هذه الحالة سيطلع

My Name is Ali and my age is 50

مثال ٢ :

```
Console.WriteLine(" x = {2} , y = {0} and z = {1} " , 10 , 20 , 30 ) ;
```

سيطلع

x = 30 , y = 10 and z = 20

٢- Concatinatin (الدمج)

```
Console.WriteLine(" Hello World " + 5 +20);
```

لكن هذه الطريقة ثابتة وماهو مكتوب أولاً سيعرض أولاً بعكس الطريقة السابقة انت من يتحكم فيما سيظهر أولاً وسيعرض الكل .

سيكون الناتج....

Hello World 520

سيعرضها بهذا الشكل ولن يجمع الرقمين فهذه الأداة (+) بعد النص في الطباعة ليس أداة جمع إنما هي أداة للدمج بين النصوص فهو يتعامل مع الأرقام التي بعد النص على أنها نص وليست رقم فمنطقياً لا يمكن إضافة رقم على نص، لكن إذا أتى الرقمان قبل جملة الطباعة فسيجمع الرقمين أولاً ويقوم بالعملية الحسابية بطريقة عادية جداً.

```
Console.WriteLine( 5 + 20 + "Hello World" );
```

سيطبع : 25Hello World

وعندما ننتهي من الكود ونريد تشغيل البرنامج نضغط Ctrl+F5.

توجد بعض العلامات تسمى " Escape Sequence " التي قد تساعدك في عملية الطباعة لتخرج النص بشكل منسق أو إنذار عند عملية ما خطأ.

طباعة علامة تنصيص فردية	'\
طباعة علامة تنصيص زوجية	"\"
طباعة علامة \	\\
لعمل إنذار	\a
للنزول إلى سطر جديد	\n
لعمل tab (مسافات فارغة)	\t

مثال :

```
Console.WriteLine(" \' Hello " );  
Console.WriteLine(" \" Hello " );  
Console.WriteLine(" \\ Hello " );  
Console.WriteLine(" \a Hello " );  
Console.WriteLine(" \n Hello " );  
Console.WriteLine(" \t Hello " );
```

4

أنواع البيانات (Data Types)

البيانات التي تُخزن في الذاكرة يجب أن يكون لها نوع حتى يتعرف عليها مترجم اللغة، فمثلاً تكون نصوص أو حرف أو عدد صحيح أو عدد عشري ويجب أن نحدد نوعها أولاً وإلا لن يتعرف عليها المترجم وسيحدث خطأ، يوجد العديد من الأنواع ولكن بالتأكيد لن نستخدم كل شيء، فقط ما تحتاج إليه في برنامجك...

نوع البيانات	النوع الموافق في منصة الدوت نت	الحجم	الوصف
bool	System.Boolean	true أو false	قيمة منطقية صح أو خطأ
sbyte	System.Sbyte	من - ١٢٨ الي ١٢٧	رقم صحيح موجب أو سالب مساحته ٨ بت
byte	System.Byte	٠ الي ٢٥٥	رقم صحيح موجب مساحته ٨ بت
short	System.Int16	-٣٢,٧٦٨ الي ٣٢,٧٦٧	رقم صحيح سالب أو موجب مساحته ١٦ بت
ushort	System.UInt16	٠ الي ٦٥,٥٣٥	رقم صحيح موجب مساحته ١٦ بت

رقم صحيح سالب أو موجب مساحته ٣٢ بت	٢,١٤٧,٤٨٣,٦٤٨- ٢,١٤٧,٤٨٣,٦٤٧	System.Int32	int
رقم صحيح موجب مساحته ٣٢ بت	٠ الى ٤,٢٩٤,٩٦٧,٢٩٥	System.UInt32	uint
رقم صحيح موجب أو سالب مساحته ٦٤ بت	- ٩,٢٢٣,٣٧٢,٠٣٦,٨٥٤,٧٧٥,٨٠ ٨ الى ٩,٢٢٣,٣٧٢,٠٣٦,٨٥٤,٧٧٥,٨٠ ٧	System.Int64	long
رقم صحيح موجب مساحته ٦٤ بت	٠ الى ١٨,٤٤٦,٧٤٤,٠٧٣,٧٠٩,٥٥١ ٦١٥	System.UInt64	ulong
يقبل حرف مساحته ١٦ بت	U+0000 الى U+ffff	System.Char	char
يقبل رقم كسري مساحته ٣٢ بت	٠ الى 1.844674407370955e+16	System.Single	float
يقبل رقم كسري مساحته ٦٤ بت	E+38 ٣.٤٠٢٨٢٢٣- الى 3.402823E+38	System.Double	double
يقبل رقم سالب أو موجب مساحته ١٢٨ بت	- E+308 ١.٧٩٧٦٩٣١٣٤٨٦٢٣٢ الى E+308 ١.٧٩٧٦٩٣١٣٤٨٦٢٣٢	System.Decimal 1	decimal
يمثل سلسلة نصية من جدول اليونكود	مقيد بحسب الذاكرة	System.String	string
	يقبل أي نوع من الأنواع السابقة	System.Object	object

إذا مما تتكون البيانات التي تُخزن في الذاكرة؟

تتكون من (نوع البيانات + المتغير الذي يحتوي على البيانات + القيمة)

Data type + variable + value

5

المتغيرات

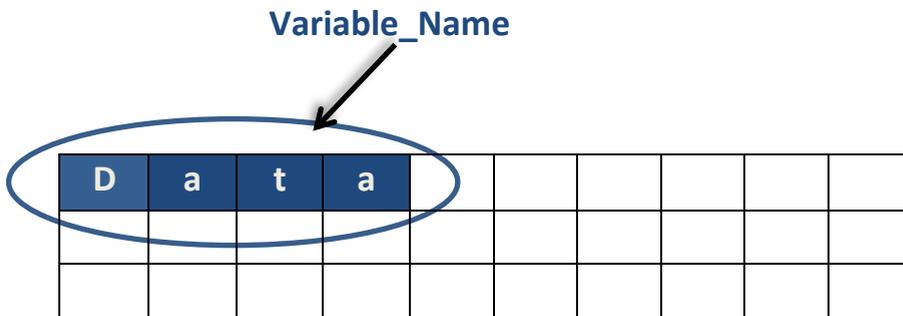
(Variables)

المتغير هو عبارة عن اسم لمكان تخزين البيانات في الذاكرة، والذاكرة عبارة عن خلايا وكل خلية تُعادل ١ بت، وتُحدّد المساحة التي تأخذها البيانات في الذاكرة من خلال نوعها وتتغير قيمة المتغير باستمرار كلما أدخلنا له قيمة جديدة، كما يمكننا تسمية المتغير بأي اسم نريده لكن لا نستطيع تسمية متغيرين بنفس الأسم، ويجب أن تكون القيمة التي ستُخزن تتناسب مع نوع المتغير.

تتم عملية تعريف المتغير بهذا الشكل :

القيمة = اسم المتغير + نوع البيانات التي تخزن في المتغير

Data type Variable_Name = value ;



وُسمي هذا المكان في الذاكرة لكي نستطيع الوصول إلى البيانات مرة أخرى بسهولة، فهو بمثابة صندوق توضع به هذه البيانات .

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int num = 5;
            string name = "123Mahmoud";
            float num2 = 34.5f;
            char c = 'a';
            double num3 = 34.5;
        }
    }
}

```

في هذا المثال عرفنا متغير من النوع (int) ووضعنا به القيمة 5

ومتغير من النوع (string) ووضعنا به نص داخل علامتين تنصيص ويجب أن يوضع النص أو الكلمة بين علامتين تنصيص، حتى إذا كانت القيمة عبارة عن أرقام لكنها بداخل علامتين تنصيص فإنه يتعامل معها على أنها نص وليست أرقام.

ومتغير من النوع (float) رقم يحتوي على علامة عشرية ونكتب في آخره (f) لكي يتعرف المترجم على أنه float، لأن القيمة الافتراضية للأعداد الكسرية هي double.

ثم عرفنا متغير من النوع (char) يأخذ حرف ويوضع بين علامة تنصيص فردية.

يجب أن نعلم أن لغة السي شارب حساسة من ناحية الحروف أي أن A لا تساوي a

فيجب أن ننتبه عند الكتابة.

✘ Int : خطأ

✓ int : صح

فمثلاً إذا كان :

```
int a=7;
```

```
int A =7;
```

فإنهما في هذه الحالة متغيران مختلفان، ويجب مراعاة الدقة في كتابة نوع المتغير والتعامل معه بشكل صحيح وإلا سيحدث خطأ في البرنامج ولن يتعرف على نوع المتغير.

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int num1=5 ;
            int num2=10 ;
            int sum ;
            sum = num1 + num2 ;
            Console.WriteLine("sum of two numbers = "
                               +sum);
        }
    }
}
```

وإذا كنت تريد تعريف أكثر من متغير من نفس النوع فمن الممكن أن نعرفه بهذه الطريقة...

```
int num1=5 , num2=10 , sum ;
```

مثال :

```
static void Main( )
{
    int x = 10 ;
    x = 5 ;
    x = 20 ;
}
```

أدخلنا ثلاث قيم للمتغير، تكون قيمة المتغير الجديدة هي آخر قيمة دخلت له. إذاً تكون قيمة المتغير x تساوي ٢٠.

ملاحظة :

لا يجوز تعريف المتغير أكثر من مرة في نفس المكان من الكود.

فمثلاً :

```
static void Main( )
{
    int x =1 ;
    int x =5 ;
}
```

خطأ لأن هذا المتغير مُعرف مُسبقاً.

تسمية المتغير

يجب أن يكون اسم معبر عن نوع البيانات التي ستُخزن فمثلاً إذا كان اسم او عُمر شخص.

```
string Name;
int Age;
```

ولا يجوز أن يبدأ المتغير بأي من الرموز الآتية :

! # \$ % ^ & - = +) (?

لكن ممكن أن يبدأ بـ _

int _name ; → صحيح

يمكننا أن نسمي المتغير بأي اسم نريده، لكن هناك بعض الكلمات المحجوزة في اللغة التي لا نستطيع تسمية المتغير بنفس اسمها ...

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit

in	in (generic modifier)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	using

			static
virtual	void	volatile	while
add	alias	ascending	
async	await	descending	
dynamic	from	get	
global	group	into	
join	let	nameof	
orderby	partial (type)	partial (method)	
remove	select	set	
value	var	when (filter condition)	
where (generic type constraint)	where (query clause)	yield	

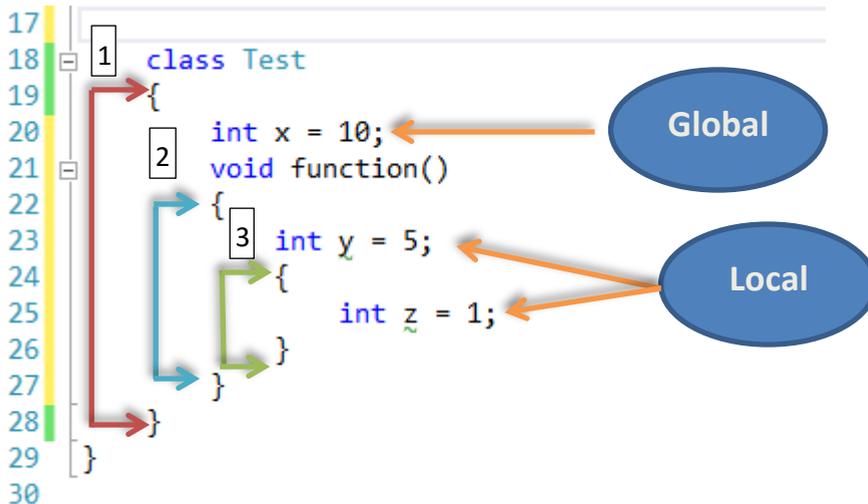
فمثلاً لا نستطيع تعريف متغير هكذا `int void` هذا غير صحيح وسيحدث خطأ، يجب الابتعاد عن هذه الكلمات في تسمية المتغيرات فهي كلمات محجوزة في اللغة ولها استخدامات معينة.

كل متغير نعرفه يكون له نطاق ومجال يُرى فيه في الكود فإما أن يكون :

Local (محلي) : ويكون ذلك في نطاق محدد ونحن نحدد مدى رؤية مترجم اللغة له في باقي الكود من خلال قوسين المجموعة `{ }`

Global (شامل أو عام) : ويكون مرئي في كل الكود داخل الـ `class` أي يكون في بداية الـ `class`.

ملاحظة : عندما تعطي قيمة ابتدائية للمتغير فإنه يستخدمها فقط في حالة أنك لم تُدخل له قيمة أخرى.
مثال :



في هذه الصورة يكون المتغير x عام، أي مرئي في كل الكود الذي في هذا الـ `class` الذي يُسمى `Test`

فإذا كُتِبَ حرف x في أي مكان في الصورة تحت اسم الـ `class` سيظهر لك المتغير x لكن بشرط أن يكون داخل مجال الـ `class` أي أنه لا يقع بين خارج أول قوس وآخر قوس للـ `class` كما يشير السهم (1) في الصورة. ولا يجوز إعادة تعريف متغير آخر بنفس اسم المتغير x لأنه مُعرَّف مُسبقاً حيث أنه متغير عام داخل الـ `class`.

أما المتغير y فإنه أقل شمولية من x حيث أنه لا يُتعرَّف عليه خارج قوسين المجموعة للدالة، يُرى بأي مكان داخل قوسين المجموعة الخاصين بالدالة كما يشير السهم (2). ويجوز تعريف متغير آخر بنفس اسمه خارج أقواس الدالة الخاصة به.

المتغير z هو الأقل شمولية فيهم حيث أنه لا يُتعرَّف عليه خارج القوسين المحيطين به كما يشير السهم (3).

كل قوس مجموعة هو عبارة عن مرحلة ومستوى كلما صعد المتغير أعلى خارج الأقواس كلما زادت شموليته ومعرفته في أجزاء أخرى في الكود.

الثابت (Constant)

الفرق بين الثابت والمتغير أن الثابت قيمته ثابتة لن تتغير طوال البرنامج. أما المتغير فيمكن تغيير قيمته، إذا حاولنا تغيير الثابت سيحدث خطأ في البرنامج.

ويكون بالشكل الآتي :

```
const + data type + name + value ;
const string name="Mahmoud" ;
```

```
const int number = 1 ;
```

ويمكن أن نستخدمه في القيم الرياضية والفيزيائية الثابتة.

تمارين ٢ :

أحسب مساحة دائرة نصف قطرها ٤

```
using System;
namespace Second_program
{
    class Program
    {
        static void Main()
        {
            const float pi = 3.14f ;
            int radius = 4 ;
            float area = pi * (radius * radius);
            Console.WriteLine(area);
        }
    }
}
```

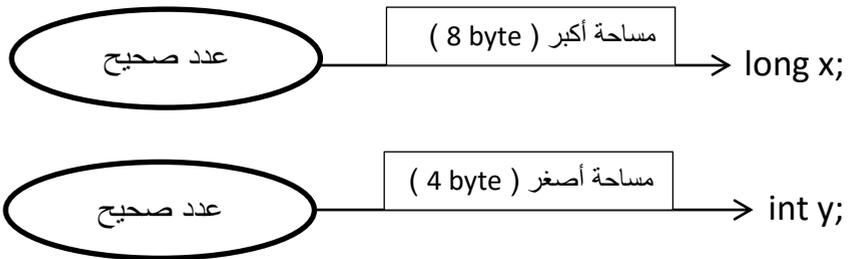
6

تحويل أنواع المتغيرات (Conversions)

١- تحويل ضمني (implicit)

وهي أن تأخذ متغير ما لتضعه في متغير أكبر منه في الحجم أو يساويه ونفس نوع البيانات بدون أن يحدث أي فقدان أو إضاعة لجزء من البيانات التي بداخله.

مثال :

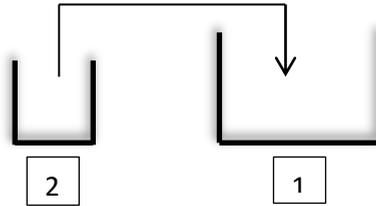


$x = y ;$

في هذه الحالة وضعنا قيمة المتغير y داخل المتغير x .

حيث أن حجم المتغير (x) ٦٤ بت وحجم المتغير (y) ٣٢ بت فيمكن للمتغير ذو الحجم الأكبر أن يحتوي على المتغير ذو الحجم الأقل بدون أي حدوث عملية فقد للبيانات. أما إذا حاولت فعل العكس في الحالة الطبيعية فسيحدث خطأ ويحدث فقد للبيانات.

إذا حاولت وضع الصندوق في الشكل (2) في الصندوق في الشكل (1) لن يحدث أي خطأ لأن الصندوق في الشكل (1) أكبر من الصندوق الآخر، لكن في حالة عكس هذه العملية سيحدث خطأ لأن الصندوق في الشكل (2) أصغر في الحجم.



٢- تحويل صريح (Explicit or Casting)

وهي عكس الحالة السابقة، وتحدث عندما يوجد احتمال فقدان في البيانات أثناء عملية تمرير البيانات إلى المتغيرات الجديدة، ويمكننا وضع بيانات المتغيرات ذي الاحجام المختلفة والانواع المختلفة داخل بعضها بشرط تغيير نوع المتغير الضيف إلى نفس نوع المتغير المُضيف.

وفي حالة الأعداد يمكننا وضع القيم الكسرية float أو double بداخل متغيرات نوعها int أو long بشرط تغيير نوعها .

مثال ١:

```
long x;
```

```
int y;
```

```
y = ( int ) x;
```

مثال ٢:

```
int x = 10 ;
```

```
float y = 3.0f ;
```

```
x = ( int ) y ;
```

أو يمكننا استخدام هذه الطريقة :

```
x = Convert.ToInt32( y );
```

وفي هذه الحالة فإن المتغير المُضيف يستطيع إستقبال المتغير الضيف.

الفرق بين :

Convert.ToInt64	Convert.ToInt32	Convert.ToInt16
تحويل المتغير إلى النوع long حيث أن النوع long حجمه 64 بت أي ٨ بايت	تحويل المتغير إلى النوع int حيث أن النوع int حجمه ٣٢ بت أي ٤ بايت	أي أنها تحول المتغير إلى النوع short حيث أن النوع short حجمه ١٦ بت أي ٢ بايت

تذكر أن :

إذا جاء حرف u قبل هذه الأنواع من البيانات

(short , int , long , float , double)

معنى ذلك أن هذه الأنواع تستطيع أخذ قيمة موجبة فقط، وحرف u إختصار لـ

.unsigned

توجد بعض الدوال لتحويل النصوص ...

مثل :

<code>decimal</code>	<u>ToDecimal(String)</u>
<code>float</code>	<u>ToSingle(String)</u>
<code>double</code>	<u>ToDouble(String)</u>
<code>short</code>	<u>ToInt16(String)</u>
<code>int</code>	<u>ToInt32(String)</u>
<code>long</code>	<u>ToInt64(String)</u>
<code>ushort</code>	<u>ToUInt16(String)</u>
<code>uint</code>	<u>ToUInt32(String)</u>
<code>ulong</code>	<u>ToUInt64(String)</u>

وتوجد أيضا الدالة <--- Parse()

لتحويل أي شيء من النوع (string)

فمثلاً إذا أردنا إدخال قيمة لمتغير نوعه int من خلال المستخدم :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int num;
            num = int.Parse(Console.ReadLine( ));
        }
    }
}
```

هذا الكود معناه أنه يوجد متغير من النوع (int) وسيأخذ قيمته من المستخدم بواسطة الدالة (ReadLine()) لكن هذه الدالة لا تقبل إلا نصوص، لذا سنجعلها تقبل أعداد صحيحة من خلال دالة (Parse()).

وأخراً سنضع كل هذا داخل المتغير الذي نريد إدخال القيمة له كما في المثال السابق.

أو

```
num = Convert.ToInt32(Console.ReadLine( ) );
```

7

التعليقات

(Comments)

التعليقات هي عبارة عن شرح موجز لجزء معين من الكود البرمجي فيجعل الكود سهل القراءة والفهم، ويتجاهل المترجم التعليقات ولا يتعامل معها نهائياً فهي مجرد ملاحظات أو تنويه يكتبه المبرمج لنفسه أو لمن يعمل بعده، إذا وُجِدَ فيها خطأ نحوي أو منطقي فلا يهم فلن يحدث أي خطأ.

أنواع التعليقات :

تعليق لسطر فردي (//)

مثال :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int num; // this is the first variable
        }
    }
}
```

تعليق متعدد الأسطر (/* */)

أي أنه يمكن أن نكتب ملاحظات في أكثر من سطر، فيكون هذا الجزء من الكود غير مرئي بالنسبة للمترجم.

مثال :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main()
        {
            int num;
            /* this is the first variable
            this is the first variable
            this is the first variable
            this is the first variable */
        }
    }
}
```

سيجاهله المترجم تماماً.

أيضا لا يجب أن ننشغل بالتعليقات كثيراً، تُكتب التعليقات في الاجزاء المهمة التي تحتاج إلى توضيح فقط.

8

المعاملات الرياضية والمنطقية

(Arithmetic and Logic operators)

المعاملات الرياضية

الجمع (+)

الطرح (-)

الضرب (*)

القسمة (/)

باقي القسمة (%)

الزيادة للمتغير بمقدار ١ (++)

النقصان للمتغير بمقدار ١ (--)

مثال :

بفرض أن

A = 5 and B = 2

A + B = 7

A - B = 3

A * B = 10

A / B = 2.5

A % B = 1

A++ = 6

A-- = 4

يمكن أن نضع علامتين الجمع بعد المتغير أو قبله.

لكن انتبه...!

++A لا تساوي A++

لأنه عندما نضعهم قبل المتغير فإن الزيادة في مقدار المتغير تحدث أولاً ثم تحدث العملية التي تليها على المتغير، أما إذا وضعنا العلامتين بعد المتغير فإن العملية ستحدث أولاً ثم بعد ذلك تحدث الزيادة في القيمة.

مثال :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main()
        {
            int num1 = 10;
            int num2 = 8;
            int sum;
            sum=(num1++) + ( ++num2);
            Console.WriteLine(sum);
            //result=19
        }
    }
}
```

```

        Console.WriteLine(num1);
        //result =11
        Console.WriteLine(num2);
        //result =9
    }
}
}

```

سيطبع ١٩ فقط، لماذا؟

أول متغير لم يحدث له تغيير ولم تزد قيمته بعد، لأنه يقوم بالعملية على المتغير أولاً ثم الزيادة ثانياً، فقام بعملية الجمع وبعدها زادت قيمته بمقدار ١، أما المتغير الثاني فزادت قيمته أولاً ثم قام بعملية الجمع، وإذا قمنا بطباعة المتغيرين بعد العملية ستجدهم قد زادا بمقدار ١، والطرح كذلك أيضاً.

مثال :

```

using System;
namespace First_program
{
    class Program
    {
        static void Main()
        {
            int A=10;
            int B=8;
            int sum;

```

```

sum= ( A++ ) + ( ++B ) + ( --A ) + ( B-- ) + ( B++ )
      + ( A ) - ( B );
// ( 10 ) + ( 9 ) + ( 10 ) + ( 9 ) + ( 8 ) + ( 10 ) - ( 9 )
Console.WriteLine( sum ); // الناتج = ٤٧
    }
}
}

```

إذا وُجِدَ متغيران نوعهما string وبينهما علامة الجمع فإنها ستندمج النصين مع بعضهما.

مثال :

```

string x = " Hello " ;
string y = " World " ;
Console.WriteLine( x + y ) ;

```

سيطبع :

Hello World

معاملات المقارنة :

يساوي (==)
لا يساوي (!=)
أكبر من (>)
أقل من (<)
أكبر من أو يساوي (>=)
أقل من أو يساوي (<=)

مثال :

بإفترض أن ...

$A = 10$, $C = 10$ and $B = 5$

$A == C$

$A > B$

$A != B$

$B < A$

$A >= C$

$B <= A$

ملاحظة :

علامة = ليست علامة يساوي إنما تسمى تعيين أو تخصيص (assignment).

فإذا قولنا أن :

$A = 5$

أو

$A = B$

معنى هذا أن قيمة الجانب الأيمن للعلامة (=) تنتقل إلى الجانب الأيسر.

القيمة 5 تنتقل إلى المتغير A

والمتغير B يوضع بداخل المتغير A وتصبح قيمة المتغير A نفس قيمة المتغير B وقيمة المتغير B لا تتأثر وتظل كما هي.

مثال :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int num1 = 10;
            int num2 = 8;
            num1 = num2;
            Console.WriteLine( num1 );
            //٨ = الناتج
            Console.WriteLine( num2 );
            // ٨ = الناتج
        }
    }
}
```

المعاملات المنطقية

Logical AND (&&) : يجب أن تكون القيمتين صحيحتين

Logical OR (||) : يجب أن تكون إحدى القيمتين صحيحة

Logical NOT (!) : يعكس القيمة، فإذا كانت صحيحة في الأصل تنفيها الأداة وتصبح خطأ والعكس.

وسنقوم بالمعنى أكثر عندما نصل إلى " الجمل الشرطية " .

المعاملات الثنائية التي تتعامل مع البت (Bit)

&

|

^

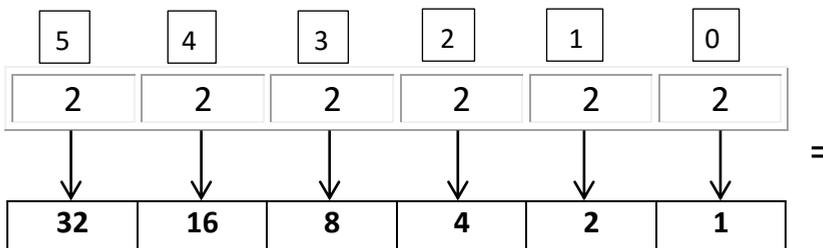
أي أنها تتعامل مع البيانات على شكل النظام الثنائي (Binary System).

ملاحظة :

أي حرف أو رقم يتم تحويله داخل الكمبيوتر إلى النظام الثنائي أي إلى أصفار وآحاد غيرها. (0111001010010110) فكما ذكرت فهذه تسمى لغة الآلة والكمبيوتر لا يفهمها.

يتم تحويل الأرقام العشرية إلى النظام الثنائي بهذه الطريقة :

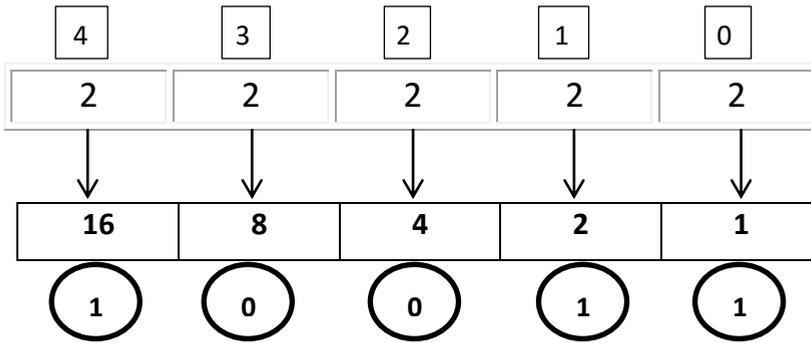
٢ أس صفر إلى ٢ أس ما لا نهاية، لذا سُميَ بالنظام الثنائي.



ويتم جمع كل القيم التي تُكوّن القيمة المُعطاة، والقيمة التي نأخذها نضع مكانها 1 والتي لم نستخدمها نضع مكانها 0. لا يشترط تكرار معين للأساس ٢، فقط ما تحتاج إليه لإخراج القيمة بالنظام الثنائي، أو يُمكنك أن تقسم القيمة المُعطاة على العدد ٢ وباقي القسمة يُكوّن القيمة بالنظام الثنائي.

مثال ١ :

حول القيمة ١٩ إلى النظام الثنائي.



ثم نبحث عن الأرقام التي تُكوّن الرقم ١٩.

فتكون (16,2,1) فنضع 1 تحت كل قيمة قد أخذناها و0 تحت كل قيمة لم نستخدمها في تكوين الرقم العشري.

أو نقسم القيمة ١٩ على ٢ ونخرج باقي القسمة

وتكون بهذا الشكل :

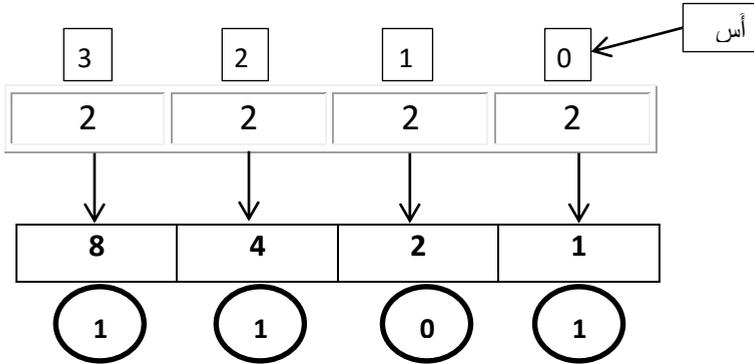
القيمة الأساس باقي القسمة

١	٢	١٩
١	٢	٩
٠	٢	٤
٠	٢	٢
١	٢	١
		١

فتكون القيمة (١٩) بالنظام الثنائي هي <--- 10011

مثال ٢ :

حول القيمة ١٣ إلى النظام الثنائي.



نبحث عن الأرقام التي تُكوّن الرقم ١٣ .

فتكون (8,4,1) فنضع 1 تحت كل قيمة قد أخذناها و 0 تحت كل قيمة لم نستخدمها في تكوين الرقم العشري.

فتكون القيمة (١٣) بالنظام الثنائي هي <--- 1101

شرح هذه الجزئية كان فقط لتفهم كيفية عمل هذه المعاملات (" & | ^ ") مع الأرقام.

مثال :

- في حالة (&)

A = 5 and B = 20

A & B =

00101

&

10100

_____ =

00100

عند جمع 1 و 0 باستخدام & فإن الناتج يكون 0، ويكون الناتج 1 في حالة واحدة... إذا كان الجمع بين 1 و 1.

إذاً $A \& B = 4$

- في حالة (|)

عند جمع 1 و 0 باستخدام | فإن الناتج يكون 1، ويكون الناتج 0 في حالة واحدة... إذا كان الجمع بين 0 و 0.

$A | B =$

00101

|

10100

=====

10101

إذاً $A | B = 21$

في حالة (^)

عند جمع 1 و 0 باستخدام ^ فإن الناتج يكون 1، ويكون الناتج 0 في حالة أن القيمتان متشابهتان 1 و 1 أو 0 و 0

$A \wedge B =$

00101

^

10100

=====

10001

إذاً $A \wedge B = 17$

A	B	A & B	A B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

معاملات النقل أو الإسناد (Assignment):

=

+=

-=

*=

/=

%=

&=

^=

|=

مثال :

$$A = 3 \text{ and } B = 2$$

$A += B$ تعني أن المتغير A ستُجمع قيمته على المتغير B ثم سيوضع ناتج الجمع بداخل A .

إنما هي إختصار لهذا النمط :

$$A = A + B$$

$$A -= B \rightarrow A = A - B$$

المتغير الثاني سيُطرح من المتغير الأول ثم ستُوضع قيمة الطرح في الجانب الأيسر.

($A *= B \rightarrow A = A * B$) حاصل ضرب القيمتين يُوضع في الجانب الأيسر.

($A /= B \rightarrow A = A / B$) حاصل قسمة القيمتين يُوضع في الجانب الأيسر.

($A \% = B \rightarrow A = A \% B$) حاصل باقي قسمة القيمتين يُوضع في الجانب الأيسر.

($A \&= B \rightarrow A = A \& B$) ناتج العملية يُوضع في الجانب الأيسر.

($A \wedge = B \rightarrow A = A \wedge B$) ناتج العملية يُوضع في الجانب الأيسر.

($A |= B \rightarrow A = A | B$) ناتج العملية يُوضع في الجانب الأيسر.

أولويات العمليات الحسابية :

() يُنْفَذُ ما بداخل الأقواس أولاً.

% / * تُحَسَبُ من الشمال إلى اليمين وأيهم يأتي أولاً فتحدث عملياته أولاً.

+ - تُحَسَبُ من الشمال إلى اليمين وأيهم يأتي أولاً فتحدث عملياته أولاً.

> < <= >= تُحَسَبُ من الشمال إلى اليمين وأيهم يأتي أولاً فتحدث عملياته أولاً.

! = == تُحَسَبُ من الشمال إلى اليمين وأيهم يأتي أولاً فتحدث عملياته أولاً.

= القيمة التي بالجانب الأيمن تنتقل للجانب الأيسر.

مثال :

$$(6 / 2 + 5) * 3 \% 7 - 1$$

الناتج : 2

$$4 / 2 + 6 * 3 - 5 \% 5 - 1 * (3 - 1)$$

الناتج : 8

توجد أيضاً بعض العوامل التي نستخدمها في الكود باستمرار مثل:
 النقطة (دوت) ← (.) تستخدم للوصول إلى الدوال والمتغيرات والـ classes
 عندما نقول ...

```
System.Console.WriteLine( );
```

System هي عبارة عن namespace و للوصول إلى ما بداخلها من classes نكتب
 نقطة (.)

Console هي عبارة عن class و للوصول إلى ما بداخله من دوال ومتغيرات نكتب
 نقطة (.)

وكذلك دالة () WriteLine و () ReadLine.

[] تُستخدم مع المصفوفات وللوصول إلى فهرس المصفوفة.

() تُستخدم مع الدوال، أي دالة يجب أن تحتوي على هذين القوسان أثناء الإنشاء
 والإستدعاء، وأيضاً نستخدمها عند تحويل أنواع البيانات.

تمارين :

أكتب برنامج ليبدّل قيمتين لمتغيرين مع بعضهما :

مثال لتبسيط الفكرة :

إذا كان معك كوب ماء وكوب آخر من الشاي و اردت أن تبديل الماء بالشاي والعكس،
 ستأتي بكوب ثالث فارغ لكي يحتوي على احد النوعين إلى أن تبديل الكوبين الآخرين.

```
using System;
namespace First_program
{
    class Program
```

```

    {
        static void Main( )
        {
            int num1=10;
            int num2=15;
            int temp;
            temp = num1;
            num1 = num2;
            num2 = temp;
            Console.WriteLine(num1);
            //num1=15
            Console.WriteLine(num2);
            //num2 =10
        }
    }
}

```

تمارين :

أكتب برنامج لحساب المتوسط الحسابي لخمس أرقام يدخلها المستخدم.

```

using System;
namespace Third_program
{
    class Program
    {
        static void Main( )
        {
            double n1 , n2 , n3 , n4 , n5 , average ;
            Console.WriteLine(" enter five numbers ");
            n1 = double.parse(Console.ReadLine( ) ) ;

```

```
n2 = double.parse(Console.ReadLine( ) );  
n3 = double.parse(Console.ReadLine( ) );  
n4 = double.parse(Console.ReadLine( ) );  
n5 = double.parse(Console.ReadLine( ) );  
average = ( n1 + n2 + n3 + n4 + n5 ) /5 ;  
Console.WriteLine( average ) ;  
    }  
    }  
}
```

9

الجمل الشرطية

(Conditional Statements)

if

هي عبارة عن قرارات قائمة على شروط، فإذا تحقق الشرط يُنفذ الكود الذي بداخل جملة التحكم (if).

مثال:

إذا كانت درجتك أكبر من ٥٠% أنت ناجح.

إذا كان تقديرك أكبر من ٨٥% فيكون امتياز.

وهكذا ...

```
if ( الشرط )
    code
```

مثال :

```
if ( grade >= 85 )
    Console.WriteLine(" Excellent " );
```

نكتب if ثم نفتح قوسين لكتابة الشرط بداخلهما ونكتب بداخلها الكود المراد تنفيذه.

وهي تنفذ سطر فردي تحتها فقط في حالة عدم وضع قوسين مجموعة.

أما إذا أردنا كتابة أكثر من أمر وأكثر من سطر نكتب الكود بداخل { }.

```
if ( your_name=="Mahmoud")
{
    Console.WriteLine("Hello Mahmoud");
    your_name="Mahmoud"+"Soliman" ;
}
```

يمكننا إجراء وتنفيذ أي كود نريده طالما هذا الشرط صحيح.

في المثال السابق كلمة grade هي عبارة عن متغير ثم الشرط >=

ومعنى الكود أنه إذا كانت درجتك أكبر من أو تساوي ٨٥ سيُنَفَّذ الكود الذي بداخل جملة if وفي هذه الحالة سيُطبع " Excellent "، فيما عدا ذلك لن يُنفذها.

ثم نكتب الكود بداخل هذين القوسان { }

ويجب كتابة الشرط بداخل هذين القوسان ()

ونستطيع كتابة أكثر من شرط بداخلها، فمثلاً إذا كنا نريد فحص بيانات شخص كاملة، مثل (اسمه ، عمره ، جنسه ، رقم تليفونه)

نحدد إذا وُجِدَ خطأ في أحد هذه الشروط يكمل تنفيذ الكود أو لا.

وذلك بواسطة :

(&& , || , ^ , !)

المعاملات المنطقية (&&) وتعني (AND) تتحقق من الجانب الأيسر والأيمن إذا وُجِدَ خطأ في احدهما فلن يتم تنفيذ أي منها ويكون ناتجها خطأ ولتعطي ناتجاً صحيحاً يجب أن يكون الجانبان صحيحين.

المعاملات المنطقية (||) وتعني (OR) تتحقق من الجانب الأيسر والأيمن ولن تعطي خطأ إلا إذا كان الجانبان شرطهما خطأ، إذا كان هناك جانب خطأ وآخر صحيح فستجاهل الخطأ وتعطي ناتج صحيح لأنه يوجد على الأقل أحدهما صحيح.

المعاملات المنطقية (^) وتعني (XOR) أي أنه لكي يتم تنفيذ الكود الذي بداخل if يجب أن يكون الجانبان مختلفين أي أن أحدهما صحيح والآخر خطأ، إذا كان ناتج الشرط في الجانبين خطأ أو كان ناتج الشرط في الجانبين صحيح فلن يتم تنفيذ الكود أيضاً. يجب أن يكونا مختلفين .

A	B	A & B	A B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

جانب أيمن && جانب أيسر

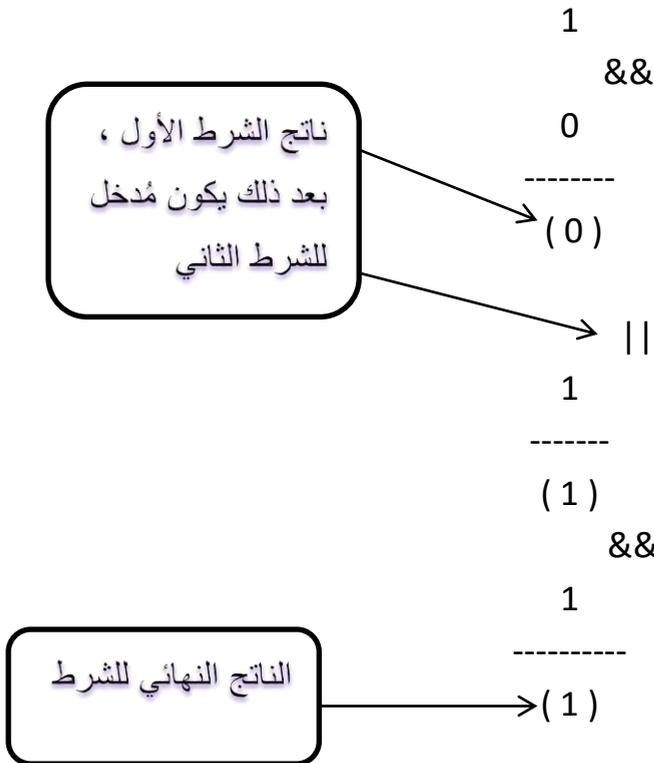
مثال :

```
if ( name == "Mahmoud" && age == 30 || gender == "Male" ||
    phone == "00000000" )
```

عندما يوجد أكثر من شرط فإنه يُقسّم كل شرطين معاً وبعد الإنتهاء من الشرطين يكون ناتجهما مُدخّل للشرط الذي يليه.

على سبيل المثال نفرض أن name قيمته صحيحة و age قيمته أيضاً صحيحة

إذا فنتاج الشرط يكون صحيحاً أما إذا كانت أحدهما خطأ فيكون ناتج الشرط خطأً، ثم هذا الناتج يكون مُدخل للشرط الذي بعده أي يكون يسار الـ (||) صحيحاً ويسارها أيضاً يكن فيكون ناتج الشرط صحيح لكليهما لأن (||) تحتاج جانب واحد صحيح على الأقل فيكون الناتج النهائي صحيحاً لهذا الشرط، يتبقى آخر فحص وهو الـ phone وناتج الشرط الذي على يساره صحيح وبما أن أداة الشرط (||) فيكون الناتج النهائي لجملة if صحيح فيتم تنفيذ ما بداخلها.



إذا كانت جملة الشرط عبارة عن متغير من النوع (bool) وكتبنا الجملة بدون وضع أي علامات تحدد الشرط وكتبنا المتغير الذي يحمل القيمة فقط فستظل هذه القيمة ثابتة فلو كانت true فإن if دائماً ستنفذ، أما إذا كانت false فلن يتم تنفيذ ما بداخل if أبداً.

مثال :

```
bool variable=false;
if( variable )
{
    Console.WriteLine(" C# ");
}
```

جملة الشرط تلقائياً تكون بهذا الشكل (variable == true) عندما يكون المتغير من النوع (bool) لو لم نكتب قيمة true أو false في الشرط.

وإذا أردنا النفي في جملة الشرط نضع علامة تعجب قبل المتغير المراد نفي قيمته (!).

```
bool variable = false ;
if( variable )
{
    Console.WriteLine(" Hello ");
}
```

في هذه الحالة لن يُنفذ الكود الذي بداخل القوسين لأن ناتج الشرط true والقيمة التي يحملها المتغير في الاصل false.

أما إذا أردت أن يتم تنفيذ الكود في حالة أن المتغير يحمل القيمة false فيجب أن تكون جملة الشرط هكذا...

```
if( ! variable )
```

أو

```
if( variable == false )
```

```

إذا كتبنا if( true)
{
    //code
}

```

في هذه الحالة سينفذ الكود الذي بداخل if لأننا حددنا أن الشرط صحيح في كل الحالات.

else

تُنفَّذ هذه الجملة عندما يكون الشرط الذي في جملة if خطأ، لكن إن كان الشرط صحيحاً فلن تُنفَّذ جملة else.

مثال :

```

int num=5;
if( num == 2 )
    Console.WriteLine("this is true");
else
    Console.WriteLine("this is false");

```

في هذه الحالة الشرط خطأ إذاً سينفذ جملة else

إذا أردنا كتابة أكثر من سطر داخلها فيجب وضع قوسين { }.

جُمْل شرط متداخلة (Nested if)

يمكننا كتابة أكثر من شرط بداخل بعضهم البعض فيمكننا كتابة أكثر من if بداخل if أو كتابة if بداخل جملة else

مثال:

```
int number = 5 ;
if ( number == 2 )
{
    // الشرط خطأ فلن يُنفذ وسيُنفذ ما بداخل else
    Console.WriteLine("this is true");
}
else
{
    if ( number > 1 )
    {
        number += 3 ;
        if ( number < 10 )
        {
            number += 5 ;
        }
        else
        {
            number -= 10 ;
        }
    }
    Console.WriteLine(number);
}
// ١٣ سيُطبع
```

معنى هذا الكود أنه يفحص قيمة المتغير أولاً فإذا كانت ٢ سيطلع `this is true` إذا كانت غير ذلك فإنه يدخل إلى `else` فيجد بداخلها جملة شرط أخرى ويفحص القيمة التي بداخل المتغير هل القيمة أكبر من ١؟

إذا كانت كذلك فيدخل إلى `if` وتزيد قيمة المتغير بمقدار ٣ ثم يفحص المتغير مرة أخرى هل قيمته أقل من ١٠؟ إذا كانت أقل فينفذ ما بداخلها وتزيد قيمة المتغير بمقدار ٥ ولو لم تكن قيمته أقل من ١٠ فلن ينفذ جملة `if` هذه وينزل إلى جملة `else` وتنقص قيمة المتغير بمقدار ١٠ وبعد ذلك يطبع قيمة المتغير في أول جملة `else` التي بداخلها كل جُمل الشرط هذه.

ifelse if

يمكننا إختبار أكثر من شرط معاً من خلال else if

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            Console.WriteLine("enter your grade");
            int grade = int.Parse(Console.ReadLine( ));
            if ( grade == 50 )
                Console.WriteLine(" your grade=50");
            else if ( grade == 60 )
                Console.WriteLine("your grade=60");
            else if ( grade == 70 )
                Console.WriteLine("your grade=70");
            else if ( grade == 80 )
                Console.WriteLine("your grade=80");
            else if ( grade == 90 )
                Console.WriteLine("your grade=90");
            else
```

```

        {
            Console.WriteLine("you didn't success if
your grade less than 50 !");
        }
    }
}
}
}

```

سيفحص أول متغير يُدخله المستخدم فإذا تحقق أول شرط سينفذه ولن ينفذ أي شيء آخر وإذا لم يتحقق سيتجه إلى الشرط الذي يليه ثم الذي يليه إلا أن يتحقق الشرط، إذا كانت كل الشروط وكل الاحتمالات خطأ سيتجه إلى else وينفذ ما بداخلها.

تمارين :

اكتب برنامج يفحص الرقم الذي يُدخله المستخدم ويعرض نوع الرقم إذا كان عدداً زوجياً أم فردياً.

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int number = int.parse(Console.ReadLine( ));
            if( number%2 == 0 )
            {
                Console.WriteLine("this number is Even");
            }
        }
    }
}

```

```

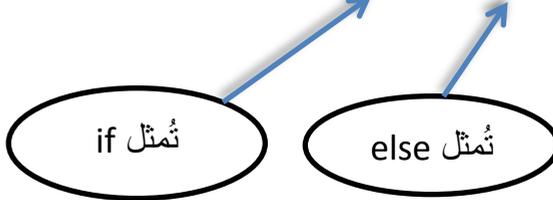
else
{
    Console.WriteLine("this number is Odd");
}
}
}
}

```

Inline Condition

يوجد نوع آخر من الجمل الشرطية وتكتب في سطر واحد وتعمل عمل if :

خطأ : صحيح ؟ (الشرط)



مثال :

```
int x =10 ;
```

```
( x > 5 && x <20 ) ? " true" : " false " ;
```

هذا الكود معناه أن في حالة أن x أكبر من ٥ وأقل من ٢٠ أطلع true ما عدا ذلك أطلع false.

ويمكننا إستقبال القيمة الناتجة من هذه العملية في متغير من نفس نوع البيانات الخارجة، في هذا الشرط تكون النواتج من النوع string فيمكننا إستقبالها في متغير نوعه string

فيمكن أن تكون بهذا الشكل :

```
string str = ( x > 5 && x < 20 ) ? " True " : " False " ;
```

```
Console.WriteLine( str );
```

النتاج :

True

أيا كان الناتج true أو false سيُخزن بداخل المتغير str.

يمكننا كتابة أي جملة نريد تنفيذها لا يشترط جملة طباعة.

مثال ٢ :

```
int num = 20 ;
```

```
num = ( num > 10 && num < 35 ) ? num +5 : num -1 ;
```

```
Console.WriteLine( num );
```

النتاج :

25

switch

هي جملة شرطية تُستخدم للتأكد من المُدخلات ما إذا كانت صحيحة أم خاطئة مثل if لكن هناك بعض الفروق البسيطة بينهما :

If	Switch
تستطيع التعامل مع أكثر من عملية بداخل الشرط الواحد .	لا تستطيع التعامل إلا مع عملية واحدة بداخل الشرط لا نستطيع استخدام == < أو > أو & % بداخلها فهي محددة .
تكون صعبة في عمليات القراءة وفهم الكود في حالة الشروط الكثيرة جداً وعند استخدام كم كبير من الأرقام للفحص عنها يُفضل استخدام switch	تكون سهلة في عملية القراءة والفهم للكود أكثر من if
يفضل استخدامها في حالة العمليات البسيطة والتي لا تحتوي على أشياء محددة فهي شمولية أكثر ، على كلٍ هي الشائعة في الاستخدام للجمل الشرطية .	أسرع من if في حالة استخدام الأرقام المحددة لأنها تتوجه مباشرة إلى الشرط الصحيح وتنفذه .

وتكون صيغتها بهذا الشكل ...

```
switch( variable)
```

```
{
```

```
case قيمة :
```

```
statement;
```

```
break; تستخدم لتوقف الفحص عندما يتحقق
```

```
الشرط
```

```
case قيمة :
```

```
statement; الجملة المراد تنفيذها إذا كان  
الشرط صحيح  
break;  
default:  
تُنفذ هذه الجملة في حالة عدم وجود شرط صحيح  
تُشبهه else في جملة if  
statement;  
الجملة المراد تنفيذها إذا كان الشرط صحيح  
break;  
}
```

مثال :

```
int x = 10 ;  
switch( x )  
{  
case 5:  
    Console.WriteLine(" this is number 5 ");  
    break;  
case 3:  
    Console.WriteLine(" this is number 3 ");  
    break;  
case 10:  
    Console.WriteLine(" this is number 10 ");  
    break;  
case 2:  
    Console.WriteLine(" this is number 2 ");  
    break;  
default:
```

```

    Console.WriteLine(" false ");
    break;
}

```

يمكننا الإستغناء عن كلمة break في بعض الحالات.

إذا كنا نريد الكشف عن أكثر من قيمة في نفس الحالة، مثلا عند القيمة كذا أو كذا... نفذ.

مثال:

أكتب برنامج للكشف عن تقدير الطلبة، يُدخله المستخدم ويطلع له التقدير الكلي .

```

int grade=int.parse(Console.ReadLine());
switch(grade)
{
case 50:
    Console.WriteLine("Accepted");
    break;
case 65:
    Console.WriteLine("Good");
    break;
case 76:
    Console.WriteLine("Very Good");
    break;
case 85:// لا يجب كتابة أي كود في هذه الحالة
case 95:
    Console.WriteLine("Excellent");
    break;

default:
    Console.WriteLine("Didn't pass");

```

```
break;
}
```

معنى الكود أن :

في حالة إذا كان التقدير ٥٠ سيطلع مقبول وإذا كان ٨٥ أو ٩٥ سيطلع امتياز

هذه الكود غير منطقي ويُفضل أن يُكتَب بـ `if`.

كان هذا للتوضيح فقط.

لكن في جملة الـ `default` يجب علينا كتابة `break`.

إذ لم تُكتب في آخر جملة سيحدث خطأ في الكود.

لأن عند عدم كتابتها فهذا معناه أنه في هذه الحالة أو التي تليها "إذا كان الشرط صحيح" نفذ. فيحدث خطأ لأنه يتوقع وجود حالة بعدها يكشف عنها ، فكيف يُنفذ ولا توجد حالة بعدها؟!!

إذاً فلا يجوز تجاهلها ويجب كتابتها في آخر جملة.

10

حلقات التكرار

(Loop Statements)

هي عبارة عن تكرار جزء معين من الكود لعدد معين من المرات، وهي مهمة جداً لأنها توفر الوقت والمجهود، تنفيذ هذه الجملة يعتمد على شرط وطالما الشرط يتحقق فهي تُنفَّذ وتنتهي ثم تبدأ مرة أخرى إلى أن تصل إلى نهاية الشرط، ويتم استخدامها عن طريق بعض الجمل التي تستخدم في عملية التكرار.

إحدى أشكال الحلقات هي :

while

وتكون جملة التكرار بالشكل التالي :

while (شرط)

{

الكود المراد تكراره

}

ومعنى الكود أنه طالما أن هذا الشرط يتحقق نفذ.

مثال :

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int i = 1;
            while( i < 10 )
            {
                Console.WriteLine( i );
            }
        }
    }
}

```

لكن إذا كتبناها بدون عملية الزيادة فستظل تعمل بدون توقف لأن الشرط دائما صحيح طالما القيمة لن تتغير، وستكون حلقة لا نهائية .

فيجب أن نزود قيمة المتغير بالمقدار الذي نريده، فمثلا إذا قولنا

`i++` سيزيد بمقدار ١

`i+=2` سيزيد بمقدار ٢

`i+=3` سيزيد بمقدار ٣

وهكذا.....

```

        }
    }
}

```

لنكتبه بصورة صحيحة :

```

using System;
namespace First_program
{
    class Program
    {

```

```

static void Main( )
{
    int i = 1;
    while( i < 10 )
    {
        Console.Write( i ) ;
        i++;
    }
}
}
}

```

سيطبع :

1 2 3 4 5 6 7 8 9

ثم سيتوقف عمل الحلقة لأنها وصلت لنهاية الشرط، حيث أن الشرط لن يتحقق لأن ١٠ ليست أقل من ١٠ عندها سيتوقف الشرط وتتوقف حلقة التكرار.

يمكننا وضع الشرط الذي نريده وبالزيادة التي نريد وكتابة أي كود بداخل جملة التكرار وسيتم تنفيذه طالما أن الشرط صحيح.

إذاً تتكون جملة التكرار من :

قيمة ابتدائية للمتغير + while (الشرط) + مقدار الزيادة (العداد).

تمارين :

أكتب برنامج يطبع حروف الهجاء من A : Z

```

using System;
namespace First_program
{

```

```
class Program
{
    static void Main( )
    {
        char alphapet = 'A';
        while( alphapet<= 'Z' )
        {
            Console.WriteLine(alphapet);
            alphapet++;
        }
    }
}
```

أو يمكننا كتابتها بهذا الشكل :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            char alphapet = 65;
            while(alphapet<= 90)
            {
                Console.WriteLine(alphapet);
                alphapet++;
            }
        }
    }
}
```

هذا في حالة إذا أردنا حروف الهجاء في حالة الحروف الكبيرة أما إذا أردناها في حالة الحروف الصغيرة سنبدأ بـ ٩٧ ونضع الشرط أقل من أو يساوي ١٢٢.

سيكون الناتج :

a

b

.

.

z

do...while

وجملة التكرار هذه تختلف عن السابقة في أنها تنفذ الكود الذي بداخلها أولاً بعد ذلك تفحص الشرط وتتأكد ما إذا كان صحيحاً أم خطأ حتى إذا كان الشرط غير صحيح فإنها تنفذ الكود مرة واحدة فقط على الأقل.

وتكون بالشكل التالي:

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int i = 1;
            do
```

```

        {
            Console.WriteLine( i );
            i++;
        }
        while( i < 10 ) ;
    }
}

```

إذا تتكون جملة التكرار هذه من :
 قيمة ابتدائية للمتغير + do + مقدار الزيادة (العداد) + while (الشرط).
 تمارين :

أكتب برنامج يأخذ الأرقام والرموز والحروف ويطبع الرقم العشري المقابل لها في الـ
 .ascii

```

using System ;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main( )
        {
            char check ;
            do
            {
                Console.Write( " enter character : " ) ;
                check = char.Parse( Console.ReadLine( ) ) ;
                int decim = check ;
                Console.WriteLine( decim ) ;
            }
            while ( true ) ;
        }
    }
}

```

for

هي شكل من أشكال حلقات التكرار وهي الشائعة في الإستخدام.
وتكون بهذا الشكل :

(مقدار الزيادة أو العداد ; الشرط ; قيمة إبتدائية للعداد) for

وتُكتب الجملة في سطر واحد بخلاف الجمل السابقة التي كانت يُكتب كل جزء منها في مكان مختلف في الكود.

تُستخدم في الجُمْل التي نعرف عدد مرات تكرارها، فيفضل إستخدامها.

والفرق بينها وبين while

while تستخدم عندما لا نعرف عدد مرات التكرار.

مثال:

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            bool question=true;
            int choice;
            while(question==true)
                // يمكننا الإستغناء عن ( ==true )
                لأنها تلقائياً تكون true في الشرط إذا لم نغيرها بـ false
            {
```

```

choice=int.parse(Console.ReadLine());
switch(choice)
{
case 1:
    statement;
    break;
case 2:
    statement;
    break;
case 3:
    statement;
    break;
default:
    question=false;
    break;
}
}
}
}

```

مثال على for لتوضيح الفرق بينهما

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int number=1 ; number<=10 ; number++ )
            {

```

```

        Console.WriteLine(number);
    }
}
}
}

```

أكتب برنامج لجمع الأرقام من ١ إلى ١٠٠

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int sum = 0;
            for( int number=1 ; number<=100 ; number++ )
            {
                sum += number;
            }
            Console.WriteLine( sum );
        }
    }
}

```

سيطبع ٥٠٥٠

ويمكننا استخدام بداخلها جملة تكرار أخرى، وتكون حلقات متداخلة ويمكننا استخدام بداخلها جملة if إذا أحتجنا إليها.

يمكننا أيضا كتابة جملة for بهذا الشكل

بشرط أن نكتب القيمة الابتدائية والشرط والعداد بداخلها ويمكننا كتابة القيمة الابتدائية فوقها، وإذا لم نكتب شيء وتركناها هكذا فستكون حلقة لا نهائية لا تتوقف.

```
for( ; ; )
```

مثال:

أكتب برنامج يبحث عن رقم يدخله المستخدم من بين الأرقام من ١ إلى ١٠٠٠ إذا وجده يطبع هذا الرقم.

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int search = int.Parse(Console.ReadLine( ));
            for( int i = 1 ; i<= 1000 ; i++ )
            {
                if ( i == search )
                {
                    Console.WriteLine("the number is "+i);
                }
            }
        }
    }
}
```

توجد بعض الكلمات التي تستخدم بداخل جملة التكرار مثل :

break : تستخدم لإيقاف حلقة التكرار.

continue : تستخدم لتخطي بعض الخطوات وتجاهلها وعدم تكرارها.

مثال :

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=10 ; i++ )
            {
                if( i == 4 )
                    break;
                Console.WriteLine( i );
            }
        }
    }
}
```

عندما تكون قيمة المتغير بـ ٤ سيخرج من حلقة التكرار ولن يكمل وسيطبع فقط...

١

٢

٣

أما إذا كتبنا..

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=10 ; i++ )
            {
                if( i==4 )
                    continue;
                Console.Write( i );
            }
        }
    }
}
```

هنا سيتجاهل هذه الخطوه وسيخطاها ويطلع :

1 2 3 5 6 7 8 9 10

إذاً كلمة break تُكتب مع switch وتُكتب أيضاً مع if لكن يجب أن تكون بداخل حلقة تكرار.

(حلقات تكرار مُتداخلة) Nested for

تُكتَب for بداخل for

سنفهم معناها وكيفية عملها بمثال.

مثال :

أطبع جدول الضرب من ١ إلى ١٢

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=12 ; i++ )
            {
                for( int j = 1; j<=12 ; j++ )
                {
                    Console.WriteLine( i * j );
                }
            }
        }
    }
}
```

في هذا الكود سيدخل بداخل أول for عند قيمة $i = 1$ بعدها سيجد for أخرى بداخلها أيضاً فيظل ينفذ الكود الذي بداخل for الداخلية إلى أن تنتهي أي يكون الشرط خطأ ثم

يصعد إلى for الأولى يزود قيمة i بمقدار ١ وطالما أن الشرط صحيح فإنه سيظل يكرر هذه العملية إلى أن يكون شرط for الأولى خطأ.

في المثال السابق :

عندما تكون قيمة $i=1$ سيدخل إلى for الثانية وتكون قيمة $z=1$ فيطبع ناتج ضرب z, i ثم يزود قيمة z بمقدار ١ فتكون ٢ ويضرب $i*z$ وهكذا إلى أن تكون قيمة $z=12$ فيطبع ناتج الضرب لأننا تضمنا في الشرط (أقل من أو يساوي) وبعدها تزيد قيمة $z=13$ فيكون الشرط في هذه الحالة خطأ لأن ١٣ ليست أقل من أو تساوي ١٢ .

فتتوقف هذه الحلقة ويصعد إلى for الأولى وتزيد قيمة ثم يدخل ينفذ ١٢ عملية تكرار أخرى ويظل يكرر إلى أن تكون قيمة $i = 13$ حينها سيكون الشرط خطأ وسيتوقف.

$1*1, 1*2, 1*3, 1*4, 1*5, 1*6, 1*7, 1*8, 1*9, 1*10, 1*11,$
 $1*12$

$2*1, 2*2, 2*3, 2*4, 2*5, 2*6, 2*7, 2*8, 2*9, 2*10,$
 $2*11, 2*12$

وهكذا....

أي أن كل عملية واحدة لأول for تعادلها ١٢ عملية لـ for التي بداخلها في هذا الكود.

أطبع هذا الشكل :

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

سنستخدم nested for لطباعة هذا الشكل.

for الخارجية تعبر عن عدد الصفوف و for التي بداخلها تعبر عن عدد الأعمدة، في هذا المثال يوجد ٦ أعمدة و ٦ صفوف، لا يشترط أن يكون عدد الأعمدة يساوي عدد الصفوف. نلاحظ أن الشكل يبدأ بعمود واحد ثم يزيد بمقدار ١ (" * ") في كل صف جديد.

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=6 ; i++ )
            {
                for( int j = 1 ; j<= i ; j++ )
                {
                    Console.Write( "*" );
                }
                Console.WriteLine( );
            }
        }
    }
}
```

في ذلك المثال حددنا له الشرط في جملة التكرار الثانية ألا تتكرر إلا إذا كان عدد الأعمدة أقل من أو يساوي عدد الصفوف، فمثلاً الصف الأول يطبع نجمة واحدة والصف الثاني يطبع نجمتين وفي الصف الثالث يطبع ثلاث نجوم وهكذا إلى أن تتوقف حلقة التكرار الخارجية.

أثناء الطباعة نجعل الأعمدة تُطبع بجانب بعضها حتى يخرج لي هذا الشكل أي أننا لا نطبع في دالة (WriteLine) بل نطبع داخل الحلقة الخاصة بالأعمدة بـ (Write) فقط وعندما يُنهي الشرط في الحلقة الخاصة بالأعمدة يخرج منها فيجد دالة (WriteLine) كي ينزل إلى صف جديد لأن هذا الصف قد أنتهى.

مثال آخر :

أطبع هذا الشكل .

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

نلاحظ أن عدد الأعمدة مكتمل ثم ينقص بمقدار ١ مع كل صف جديد.

يمكننا كتابة الكود بأكثر من طريقة فمثلاً يمكننا كتابته بهذا الشكل ...

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=6 ; i++ )
            {
```

```
        for( int j=i; j<= 6 ; j++ )
        {
                Console.Write( "*" );
        }
        Console.WriteLine( );
    }
}
}
```

أو بتلك الطريقة...

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<= 6 ; i++ )
            {
                for( int j = 6 ; j>= i ; j-- )
                {
                    Console.Write( "*" );
                }
                Console.WriteLine( );
            }
        }
    }
}
```

مثال :

أطبع هذا الشكل ...

```

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *

```

في هذا الشكل نحتاج إلى طباعة مسافات على الجانب الأيسر أولاً ونلاحظ أن هذه المسافات تقل بمقدار ١ مع كل صف جديد، إذاً سنطبع مسافات وأعمدة وصفوف، سنحتاج إلى for للصفوف بداخلها حلقتين تكرر أحدهما للمسافات والأخرى للأعمدة.

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=8 ; i++ )
            {
                for( int k = 1 ; k<=8 -i ; k++ )
                {
                    Console.Write( " " );
                }
                for( int j = 1 ; j<=i ; j++ )
                {

```

```

        Console.Write( "*" );
    }
    Console.WriteLine( );
}
}
}
}
}

```

مثال :

أطبع هذا الشكل...

```

      *
     ***
    *****
   *********
  ***********
 *****
*****
*****
*****
*****
*****
*****

```

هو نفس المثال السابق لكننا سنضاعف عدد النجوم فقط .

```

using System;
namespace First_program
{
    class Program

```

```
{
    static void Main( )
    {
        for( int i = 1 ; i<=8 ; i++ )
        {
            for( int k = 1 ; k<=8 - i ; k++ )
            {
                Console.Write( " " );
            }
            for( int j = 1 ; j<= i*2-1 ; j++ )
            {
                Console.Write( "*" );
            }
            Console.WriteLine( ); /*
            لكي ينزل إلى سطر جديد مع نهاية كل صف قد أكتملت دورته
            */
        }
    }
}
```

مثال :

أطبع هذا الشكل...

```

*
***
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            for( int i = 1 ; i<=12 ; i++ )
            {
                for( int k = 1 ; k <=12 - i ; k++ )
                {
                    Console.Write( " " );
                }
            }
        }
    }
}

```

```
        for( int j = 1; j <= i*2-1 ; j++ )
        {
            Console.Write( "*" );
        }
        Console.WriteLine( );
    }
for( int i = 8 ; i<=12 ; i++ )
{
    for( int k = 1 ; k <=12 - i ; k++ )
    {
        Console.Write( " " );
    }
    for( int j = 1 ; j<= i*2-1 ; j++ )
    {
        Console.Write( "*" );
    }
    Console.WriteLine( );
}
}
```

في هذا المثال أردنا طباعة مثلث متساوي الساقين لذا ضاعفنا الأعمدة، فجعلنا حلقة التكرار الخاصة بالأعمدة تدور إلى ضعف عدد الصفوف. ثم أنقصنا منها ١ ليكون رأس المثلث نجمة واحدة وليست ٢.

ثم بدأنا من منتصف الصفوف بعدما أنهينا من المثلث متساوي الساقين لتتكون الشجرة.

هذه الأمثلة فقط لتجعلك متمكن من استخدام حلقات التكرار بداخل بعضها.

foreach

تستخدم أيضاً للتكرار لكن هذه الجملة تستخدم في حالة وجود مجموعة من البيانات المرتبطة معاً ومُخزنة معاً بداخل مُجمَع من البيانات مثل المصفوفة " Array ".
ويكون تعريفها بهذا الشكل :

(المصفوفة أو مجموعة البيانات + الكلمة المحجوزة **in** + نوع البيانات) foreach

```
foreach( DataType variable in data )
{ code ; }
```

مثال:

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int[ ]arr = new int[ 5 ] { 1 , 2 , 3 , 4 , 5 };
            foreach( int i in arr )
            {
                Console.WriteLine( i );
            }
        }
    }
}
```

هذا مجرد مثال للتوضيح فقط وسأشرح المصفوفات لاحقاً.

11

المصفوفات (Arrays)

(المصفوفة الاحادية أو الفردية)

المصفوفة عبارة عن مجموعة من المتغيرات وتكون من نفس نوع البيانات وتكون متتالية في الذاكرة، أي أنها تحجز خلايا متتالية في الذاكرة المؤقتة (RAM) ويكون عنوان هذه الخلايا متتابع في الذاكرة، ويكون لها حجم ثابت أي أننا يجب أن نحدد عدد العناصر التي سنحتاج استخدامها، فائدة المصفوفة أنها تمكننا من التعامل مع أكثر من عنصر في نفس الوقت ونُعدّل ونغير من قيمته وتحدث عليه أي عملية بطريقة سهلة جداً وبسيطة، يمكننا أن نغير قيمة آلاف البيانات في سطر واحد، وذلك باستخدام المصفوفة.

تعريف المصفوفة يكون بالشكل التالي :

; [حجم المصفوفة] + نوع البيانات + new = اسم المصفوفة [] نوع البيانات .

Data Type + [] + Array_Name =new Data Type [size];

يجب أن نحدد حجم المصفوفة أثناء التعريف، وكل عنصر هو عبارة عن متغير بداخل المصفوفة.

أضيفت كلمة new بداخل السي شارب وهي تعبر عن أن المصفوفة ديناميكية أي يجوز تغيير حجمها، فعند تغيير حجمها تأخذ نسخة من المصفوفة القديمة وتنشئ

مصفوفة جديدة بنفس الحجم الذي حددته، حيث أن المصفوفة سابقاً كانت حجمها ثابت غير قابل للتعديل، أما حالياً يجوز تغيير حجمها متى أردنا.

لكي نشير إلى عنصر معين بداخل المصفوفة نكتب اسم المصفوفة وموضع العنصر بداخل المصفوفة، نكتب موضع العنصر بداخل قوسين [].

ويبدأ ترقيم المصفوفة من صفر حيث يكون أول عنصر في المصفوفة ترقيمه صفر.

مثال :

```
int [ ] arr = new int [10];
```

ويمكن أن تأخذ حجم المصفوفة من المستخدم بالطريقة العادية.

```
int size = int.Parse(Console.ReadLine( ) ) ;
```

```
int [ ] arr = new int[ size ];
```

معني هذه الجملة أننا أنشأنا مصفوفة بداخلها عشرة عناصر (متغيرات) من النوع (int)، ويمكننا إجراء أي عملية عليهم، جمع أو طرح أو قسمة أو ضرب أو أي عملية، يمكننا أن نعطي لعناصر المصفوفة قيم ابتدائية بهذه الطريقة...

```
int [ ] arr = new int [10] { 1 , 4 , 12 , 65 , 14 , 16 } ;
```

أو بهذه الطريقة مباشرةً ..

```
int [ ] arr={ 1 , 4 , 12 , 65 , 14 , 16 } ;
```

سيضع هذه القيم في أول ٦ خلايا من المصفوفة، أما باقي العناصر من المصفوفة سيضع لها قيمة افتراضية صفر.

arr[0]	1
arr[1]	4
arr[2]	12
arr[3]	65
arr[4]	14
arr[5]	16
arr[6]	0
arr[7]	0
arr[8]	0
arr[9]	0

دائماً ما يكون موضع آخر عنصر في المصفوفة = حجم المصفوفة - ١

مثال :

إذا أردنا إجراء عملية جمع على العنصر الخامس في المصفوفة ونضيف على قيمة العنصر القيمة ٢٠، إذاً سيكون العنصر رقم ٤ لأن المصفوفة تبدأ بصفر.

فتحدث العملية بالشكل التالي...

```
arr[ 4 ] += 20 ;
```

ويمكننا أيضاً إضافة عنصر بداخل المصفوفة إلى عنصر آخر بداخلها

أو نسخ قيمة عنصر إلى عنصر آخر

مثال:

```
arr[2] = arr[1];
```

معنى هذه الجملة أن قيمة العنصر الثاني في المصفوفة نُسخَت في العنصر الثالث، وأصبحت قيمة العنصر الثالث بنفس قيمة العنصر الثاني.

```
arr[5] += arr[2];
```

في هذه الجملة أُضيفت قيمة العنصر الثالث إلى قيمة العنصر السادس وُضع ناتج الجمع بداخل العنصر السادس.

```
arr[3] /= arr[4];
```

في هذه الجملة قَسَمَ قيمة العنصر الرابع على العنصر الخامس ووضع ناتج القسمة بداخل العنصر الرابع.

```
int sum = arr[2]+arr[4];
```

جُمِعَت قيمة العنصر الخامس مع قيمة العنصر الثالث وُضع ناتج الجمع بداخل المتغير .sum

arr[0] موضع العنصر الاول في المصفوفة.

arr[1] موضع العنصر الثاني في المصفوفة.

arr[2] موضع العنصر الثالث في المصفوفة.

وهكذا ...

نستطيع تغيير حجم المصفوفة بهذه الطريقة...

نفترض أننا أنشأنا مصفوفة بهذه الطريقة ونريد تغيير مساحتها :

```
int [ ] numbers = new int [ 10 ];
```

يحدث التعديل بهذا الشكل :

```
Array.Resize( ref numbers , 20 ) ;
```

كلمة Array هي عبارة عن class يحتوي بداخله على دالة تستطيع إعادة تعيين حجم المصفوفة فتنسخ محتوى المصفوفة القديمة وتُدخله في المصفوفة الجديدة، حيث تنشئ مصفوفة جديدة بالحجم الجديد.

Resize : هي دالة التعديل وإعادة تعيين حجم المصفوفة، وهي دالة جاهزة في اللغة.

ref : هي اختصار لـ reference ومعناها أن هذه التغيير سيحدث فعلياً في هذا المكان بداخل الذاكرة (RAM).

نمرر لهذه الدالة اسم المصفوفة والحجم الجديد.

ويتم التعامل مع العناصر داخل المصفوفة من خلال حلقات التكرار for أو foreach أو while.

مثال :

أنشئ مصفوفة مكونة من ١٠ أرقام و ضع قيم أولية بداخلها ثم اطبع هذه القيم.

```
using System ;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int [ ] num=new int[ 10 ] { 11 , 2 , 3 , 4 , 5 ,
                                     12 , 14 , 16 , 17 , 9 } ;
            for( int counter = 0 ; counter<num.Length ; counter++ )
```

```

        {
            Console.WriteLine( num[ counter ] );
        }
    }
}

```

length : هي خاصية تُرجع أو تحسب لنا حجم المصفوفة، كان من الممكن أن نكتب المساحة بشكل مباشر لكن من المفضل أن نكتب الجملة بهذا الشكل حتى إذا تغيرت المساحة فلن نضطر إلى تغييرها في كل الشروط المتعلقة بهذه المصفوفة، فهذه الخاصية تحسبها تلقائياً. وسنعرف ماهي الخصائص فيما بعد.

أو يمكننا استخدام **foreach** فهي أبسط في الكتابة من **for** وتأتي بنفس النتيجة.

```
foreach ( int count in num)
```

```
Console.WriteLine( count ) ;
```

أنشأنا عداد لكي يدور على عناصر المصفوفة، ثم يطبع محتوى المصفوفة.

مثال :

أنشئ مصفوفة مكونة من 15 عنصر وأجعل المستخدم يدخل قيم لهذه العناصر، ثم أضف القيمة 5 إلى قيمة العنصر العاشر وأطبع كل قيم العناصر.

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {

```

```

int [ ] num = new int[15];
for( int counter = 0 ; counter<num.Length ; counter++ )
{
    num[counter]=int.Parse(Console.ReadLine);
}
for( int counter = 0 ; counter<num.Length ; counter++ )
{
    if ( num[counter]==10 )
        num[counter]+=5;
}
for( int counter = 0 ; counter<num.Length ; counter++ )
{
    Console.WriteLine( num[counter] );
}
}
}
}

```

ملاحظة— لا يجوز أن تبدأ العداد بصفر وأن تجعله يزيد بمقدار ١ ثم تكتب في الشرط أقل من أو يساوي (قيمة حجم المصفوفة)، لأنك إذا كتبت أو يساوي فأنت تقول أن المصفوفة تحتوي على عدد العناصر الموجودة + ١ وفي هذه الحالة سيحدث خطأ لأنك إذا بدأت مصفوفة مكونة من ١٠ عناصر بصفر وانتهيت بالقيمة ١٠ سيكون عدد العناصر ١١ وليس ١٠ وسيحدث خطأ أثناء وقت التشغيل، يجب أن تبدأ بصفر وتنتهي بـ ٩

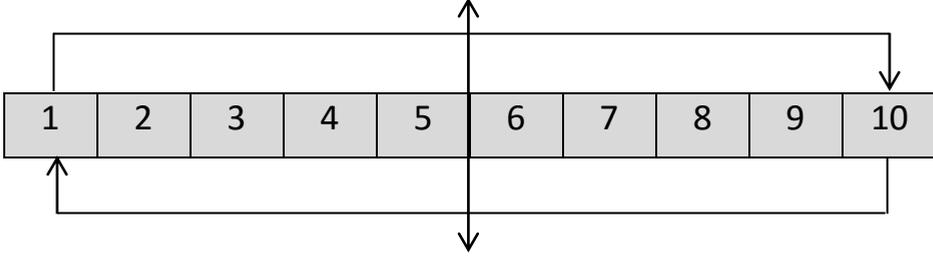
counter <= num.Length-----x (خطأ)

تمارين :

ادخل ١٠ عناصر في مصفوفة ثم اعكس هذه المصفوفة، أي أن تجعل العنصر الأخير يكون في مقدمة المصفوفة والعنصر الأول ينتقل إلى آخرها.

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int [ ] arr=new int[10];
            for( int counter = 0 ; counter<arr.Length ; counter++ )
            {
                arr[counter]=int.Parse(Console.ReadLine);
            }
            for( int counter = 0 ; counter<arr.Length/2 ; counter++ )
            {
                int temp;
                temp = arr[counter];
                arr[counter] = arr[arr.Length-1-counter];
                arr[arr.length-1-counter] = temp;
            }
        }
    }
}
```

فكرة البرنامج أننا سنجري عملية تبديل بين كل عنصرين متقابلين، وسنظل نُجري هذه العملية إلى أن نصل إلى منتصف المصفوفة، أي أنه ستحدث عملية التبديل 5 مرات وليس 10 لأننا لو أجرينا عملية التبديل 10 مرات سترجع المصفوفة إلى ما كانت عليه في السابق.



١ مع ١٠

٢ مع ٩

٣ مع ٨

٤ مع ٧

٥ مع ٦

لو جعلنا الـ for تدور إلي ما بعد منتصف المصفوفة فإنها ستبدلهم مرة أخرى.

لكن ماذا لو كان طول المصفوفة عدداً فردياً مثل 11؟

لن يؤثر ذلك في شيء لأن النوع int لا يقبل إلا أعداد صحيحة فخارج القسمة سيكون عدد صحيح وسيتجاهل الأعداد العشرية فناتج قسمة 12 على 2 يساوي 5، وسيكون في منتصف المصفوفة رقم على يمينه 5 أرقام وعلى يساره 5 أرقام، سنبدل اليمين باليسار والمنتصف لن يتأثر لأنه في كل الحالات سيظل في المنتصف.

مثال :

٣- ٢- ١- ٠ ١ ٢ ٣

إذا أردت أن تبدل الثلاثة أرقام التي على يمين الصفر بالثلاثة التي على يساره، هل سيتأثر مكان الصفر؟

الجواب لا، نفس الفكرة لن يتأثر مكان العنصر الذي في منتصف المصفوفة ذات الحجم الفردي.

وأنشأنا متغير ليحمل القيمة مؤقتاً حتى تتم عملية التبديل ووضعنا به أول عنصر ثم وضعنا آخر عنصر في مكان أول عنصر، بعدها وضعنا القيمة التي بداخل المتغير المؤقت في مكان آخر عنصر وهكذا.

مثال ٢:

انشيء مصفوفة و أدخل بها ١٠ عناصر ثم رتب العناصر تصاعدياً.

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int [ ] arr = new int[ 10 ] ;
            for( int i= 0 ; i<arr.Length ; i++ )
            {
                arr[i]=int.Parse(Console.ReadLine));
            }
            for( int i = 0 ; i <arr.Length-1 ; i++)
            {
                for( int j = i ; j <arr.Length ; j++)
```

```

        {
            if( arr[ i ] > arr[ j ] )
            {
                int temp;
                temp = arr[ i ] ;
                arr[ i ] = arr[ j ] ;
                arr[ j ] = temp ;
            }
        }
    }
}

```

فكرة هذا البرنامج أننا نقارن كل عنصر بالذي يليه فإذا كان ما يليه أقل منه فإنه سيبدل القيمتين ببعضهما.

وكما عرفنا، تتم عملية التبديل بين قيمتين عن طريق وجود متغير ثالث يحمل قيمة أحدهما حتى تتم عملية التبديل.

تمارين :

أكتب كود يجعل المستخدم يبدل بين قيمتين من خلال فهرس هذه القيم في المصفوفة، المصفوفة مكونة من ١٠ عناصر.

```

using System ;

namespace ConsoleApplication
{
    class Program

```

```
{
    static void Main ( )
    {
        int [ ] arr = new int [ 10 ] { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 };
        int index1 = int.Parse( Console.ReadLine() );
        int index2 = int.Parse( Console.ReadLine() );
        int temp ;
        for ( int i = 0 ; i < 10 ; i++ )
        {
            for ( int j = i ; j < 10 ; j++ )
            {
                if ( i == index1 && j == index2 )
                {
                    temp = arr [ index1 ] ;
                    arr [ index1 ] = arr [ index2 ] ;
                    arr [ index2 ] = temp ;
                }
            }
        }
    }
}
```

المصفوفة ثنائية الأبعاد

هي شكل من أشكال المصفوفات ويكون لها بُعدين ويتم التخزين بها عن طريق إستخدام رقم الصف والعمود.

يجب أن تحدد عدد الأعمدة والصفوف التي بها فمساحتها تساوي عدد الصفوف في عدد الأعمدة.

مثال :

	عمود 0	عمود 1	عمود 2
صف 0	12	54	6
صف 1	76	23	3
صف 2	4	1	8

هذه المصفوفة تحتوي على ٣ صفوف و ٣ أعمدة وأيضا يبدأ ترقيم الصف أو العمود من صفر.

وتتم عملية تعريف المصفوفة بهذا الشكل:

```
DataType + [ , ] + name = new DataType [ rows , cols ];
```

```
int [ , ] arr = new int [ 5 , 4 ];
```

معنى هذا الكود أننا أنشأنا مصفوفة ثنائية اسمها arr وبها ٥ صفوف و ٤ أعمدة.

فمثلاً إذا أردنا تخزين قيمة في مكان محدد في المصفوفة فإننا نحدد مكان الصف والعمود.

مثال :

في حالة تخزين القيمة 100 بداخل هذه المصفوفة، نحدد بأي صف و بأي عمود وبتلاقي الصف مع العمود نضع القيمة، تماما كالرسم البياني للمحورين (س ، ص).
لنحدد مثلا الصف الثالث والعمود الثاني.

فتكتب بهذا الشكل :

$arr[2 , 1] = 100 ;$

	عمود 0	عمود 1	عمود 2	عمود 3
صف 0				
صف 1				
صف 2		<u>100</u>		
صف 3				
صف 4				

تتم عملية الإدخال لهذه المصفوفة بتلك الطريقة :

سنفرض أن مساحة المصفوفة هي 4 صفوف و 3 أعمدة وسنحفظ بداخلها أسماء الطلاب.

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            string [ , ] arr = new string [ 4 , 3 ] ;

            for( int i = 0 ; i<4 ; i++ )
            {
                for( int j = 0 ; j<3 ; j++ )
                {
                    arr[ i , j ]= Console.ReadLine( ) ;
                }
            }
        }
    }
}

```

في عملية تعريف المصفوفة يجب أن نحدد لها عدد الصفوف والأعمدة أو أن نأخذ قيمتهما من المستخدم.

مثال :

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int row = int.Parse( Console.ReadLine( ) ) ;
            int col = int.Parse( Console.ReadLine( ) ) ;

```

```

string [ , ] arr=new string[ row , col ];
for( int i = 0 ; i< row ; i++ )
    {
        for( int j=0 ; j< col ; j++ )
            {
                arr[ i , j ] = Console.ReadLine( );
            }
        }
    }
}

```

تمارين :

أنشيء مصفوفة ٣ * ٣ لتخزين مرتبات موظفين ثم أطلع أعلى مرتب وأقل مرتب.

```

using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            int row=3;
            int col=3;
            int max_value = 0 ;
            int mini_value ;
            int [ , ] arr = new int[ row , col ];
            for( int i = 0 ; i< row ; i++ )
                {
                    for( int j = 0 ; j< col ; j++ )

```

```
        {
            arr[ i , j ]=int.Parse(Console.ReadLine());
        }
    }
    mini_value = arr[ 0 , 0 ];

    // فرضاً أن أول قيمة في المصفوفة هي أقل قيمة

    for( int i = 0 ; i< row ; i++ )
    {
        for( int j = 0 ; j< col ; j++ )
        {
            if ( arr[ i , j ] > max_value)
            {
                max_value = arr[ i , j ] ;
            }
            if ( arr[ i , j ] < mini_value)
            {
                mini_value = arr[ i , j ] ;
            }
        }
    }

    Console.WriteLine("the maximum salary="+max_value);
    Console.WriteLine("the minimum salary="+mini_value);
```

```

    }
}
}

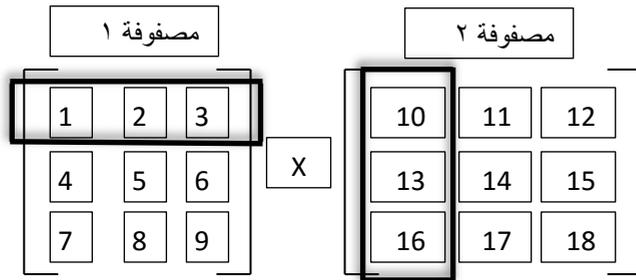
```

تمارين :

أنشيء مصفوفتين مساحتهما $3 * 3$ ثم أضرب المصفوفتين في بعضهما وخرن ناتج الضرب في مصفوفة جديدة.

ملاحظة :

عند ضرب مصفوفتين فإن كل صف في المصفوفة الأولى (١) يُضرب في كل أعمدة المصفوفة الثانية (٢).



فتضرب المصفوفتان بهذه الطريقة :

$1*10+2*13+3*16$	$1*11+2*14+3*17$	$1*12+2*15+3*18$
$4*10+5*13+6*16$	$4*11+5*14+6*17$	$4*12+5*15+6*18$
$7*10+8*13+9*16$	$7*11+8*14+9*17$	$7*12+8*15+9*18$

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main( )
        {
            int multi = 1 , temp = 0 ;
            int [ , ]arr = new int[ 3 , 3 ] ;
            int [ , ]arr2 = new int[ 3 , 3 ] ;
            int [ , ]arr3 = new int[ 3 , 3 ] ;
            Console.WriteLine("-----Enter values of array 1-----");
            for( int i = 0 ; i<3 ; i++ )
            {
                for( int j = 0 ; j<3 ; j++ )
                {
                    Console.Write(" [ " + i + " , " + j + " ] = ");
                    arr[ i , j ] = int.Parse(Console.ReadLine( ));
                }
            }
            Console.WriteLine("-----Enter values of array 2-----");
            for( int i = 0 ; i<3 ; i++ )
            {
                for( int j = 0 ; j<3 ; j++ )
                {
                    Console.Write(" [ " + i + " , " + j + " ] = ");
                    arr2[ i , j ] = int.Parse(Console.ReadLine( ));
                }
            }
            for( int k = 0 ; k<3 ; k++ )
            {
                for( int i = 0 ; i<3 ; i++ )
                {
                    for( int j = 0 ; j<3 ; j++ )
                    {
```

```

        multi = arr[ k , j ] * arr2[ j , i ];
        temp += multi;
    }
    arr3[ k , i ] = temp;
    temp = 0 ;
}
}
Console.WriteLine(" multiplication Result of array1 and
                    array2 : ");
for( int i = 0 ; i<3 ; i++)
{
    for( int j = 0 ; j<3 ; j++ )
    {
        Console.Write( arr3[ i , j ] + "\t" );
    }
    Console.WriteLine( );
}
}
}
}

```

تمارين :

أكتب برنامج باستخدام المصفوفات لتنفيذ لعبة **XO**.

X-O		
X	X	O
X	O	X
O	X	O

```
using System;
namespace ConsoleApplication7
{
    class Program
    {
        static void Main(string[] args)
        {
            const char player1 = 'x', player2 = 'o', empty = ' ';
            char[,] game = new char[3, 3];
            char winner = ' ';
            int pos1;
            int pos2;
            bool choice = true ;
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    if (choice == true)
                    {
                        Console.WriteLine(" X Playing ");
                        Console.WriteLine(" enter your positions ");
                        pos1 = int.Parse(Console.ReadLine());
                        pos2 = int.Parse(Console.ReadLine());
                        if (pos1 > 2 || pos2 > 2)
                        {
                            Console.WriteLine("\t\t wrong position");
                            j--;
                        }
                        else if (game[pos1, pos2] == 'o' || game[pos1, pos2] == 'x')
                        {
```

```
        Console.WriteLine("\t\t wrong position");
        j--;
    }
    else
    {
        game[pos1, pos2] = 'x';
        Console.WriteLine();
        choice = false ;
    }
}
else
{
    Console.WriteLine(" O Playing ");
    Console.WriteLine(" enter your positions ");
    pos1 = int.Parse(Console.ReadLine());
    pos2 = int.Parse(Console.ReadLine());
    if (pos1 > 2 || pos2 > 2)
    {
        Console.WriteLine("\t\t wrong position");
        j--;
    }
    else if (game[pos1, pos2] == 'x' || game[pos1, pos2] == 'o')
    {
        Console.WriteLine("\t\t wrong position");
        j--;
    }
    else
    {
        game[pos1, pos2] = 'o';
        Console.WriteLine( );
    }
}
```

```
        choice = true ;
    }
}
for (int a = 0; a < 3; a++)
{
    for (int b = 0; b < 3; b++)
    {
        switch (game[a, b])
        {
            case player1:
                Console.Write("X");
                break;
            case player2: Console.Write("O");
                break;
            case empty:
                Console.Write(" ");
                break;
        }
        if (b < 2)
            Console.Write(" | ");
    }
    if (a < 2)
    {
        Console.WriteLine();
        Console.WriteLine(" -----");
    }
}
Console.WriteLine(); Console.WriteLine();
for(int k=0;k<3;k++)
{
```

```
for(int g=0;g<1;g++)
{
    if (game[k , g]=='x' && game[k , g+1] == 'x'
        && game[k , g+2] == 'x')
    {
        Console.WriteLine("\t\t X Wins ");
        winner = player1;
        break;
    }
else if (game[g , k] == 'x' && game[g+1 , k]== 'x'
    && game[g+2 , k] == 'x')
    {
        Console.WriteLine("\t\t X Wins ");
        winner = player1;
        break;
    }
else if(game[0,0]=='x' && game[1,1]=='x' &&
game[2,2]=='x')
    {
        Console.WriteLine("\t\t X Wins ");
        winner = player1;
        break;
    }
else if(game[2,0]=='x' &&
    game[1,1]=='x' && game[0,2]=='x')
    {
        Console.WriteLine("\t\t X Wins ");
        winner = player1;
        break;
    }
}
```

```
    }
    if (winner == player1 || winner == player2)
        break;
}
for(int k=0;k<3;k++)
{
    for(int g=0;g<1;g++)
    {
        if(game[k,g]=='o' && game[k,g+1]=='o'
            && game[k,g+2]=='o')
        {
            Console.WriteLine("\t\t O Wins ");
            winner = player1;
            break;
        }
        else if(game[g,k]=='o' &&
game[g+1,k]=='o' && game[g+2,k]=='o')
        {
            Console.WriteLine("\t\t O Wins ");
            winner = player1;
            break;
        }
        else if(game[0,0]=='o' && game[1,1]=='o'
            && game[2,2]=='o')
        {
            Console.WriteLine("\t\t O Wins ");
            winner = player1;
            break;
        }
        else if(game[2,0]=='o' && game[1,1]=='o'
```

```
                && game[0,2]=='o')
            {
                Console.WriteLine("\t \t O Wins ");
                winner = player1;
                break;
            }
        }
        if (winner == player1 || winner == player2)
            break;
    }
    if (winner == player1 || winner == player2)
        break;
}
if (winner == player1 || winner == player2)
    break;
}
if (winner != player1 && winner != player2)
    Console.WriteLine("\t \t Draw ");
Console.WriteLine(); Console.WriteLine();

for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 3; j++)
    {
        switch (game[i, j])
        {
            case player1:
                Console.Write("X");
                break;
            case player2:
```

```
        Console.Write("o");
        break;
    case empty:
        Console.Write(" ");
        break;
    }
    if (j < 2)
        Console.Write(" | ");
    }
    if (i < 2)
    {
        Console.WriteLine( );
        Console.WriteLine("-----");
    }
    }
    Console.WriteLine( );
}
}
}
```

يُمكن تنفيذ اللعبة بشكل أبسط من هذا لكن لم نتطرق لهذه الأجزاء إلى الآن لذا فهذا كان مجرد تطبيق على المصفوفات فكل ما يهمنا هو إتقانها أكثر ومعرفة كيفية عملها.

12

النصوص

(Strings)

النصوص هي عبارة عن متسلسلة أو مصفوفة من الحروف، فيمكنك فعل ما تشاء في النص من حذف أو إضافة أو إقتصاص.

فيمكنك تعريف النص بهذه الطريقة :

```
string var_name = " value ";
```

القيمة أو النص يجب أن يُوضع بين علامتين تنصيص وبدخلها يمكنك أن تكتب أي شيء حتى إذا كانت أرقام أو مسافة خالية أو نص مدمج مع رقم فإنه سيتعامل معها على أنها نص لأنها وُضعت بين علامتين تنصيص.

مثل :

```
string name = "123 mahmoud";
```

```
Console.WriteLine( name );
```

النتاج :

```
123 mahmoud
```

ويمكنك أن تضيف أي قيمة من النوع string أو char على قيمة المتغير الذي يكون من النوع string

```
string name1 = "mahmoud" , name2="soliman", name3;
```

```
name3 = name1 + name2;
```

أو يمكننا أن نضيف النص أو الحرف الجديد على القديم بطريقة مباشرة.

مثال :

```
string name ="Mahmoud";
name += " Soliman";
Console.WriteLine( name );
```

الناتج :

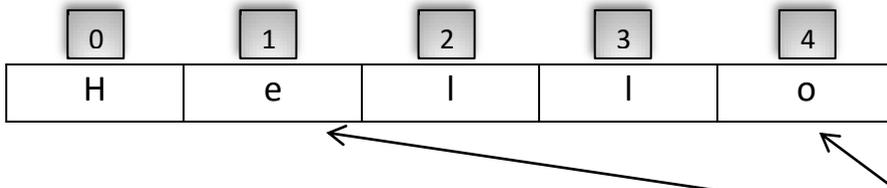
Mahmoud Soliman

ويُمكنك أن تأخذ القيم من المستخدم بالطريقة العادية.

أول حرف في النص يأخذ الترقيم (index) 0 كما في المصفوفة لأنه كما قلت هو عبارة عن مصفوفة من الحروف.

مثال :

```
string x = "Hello";
```



```
Console.WriteLine(" the result {0} {1} ", x [ 1 ] , x [ 4 ] );
```

الناتج :

e o

- لمعرفة عدد حروف نص معين نستخدم الخاصية (Length).

```
Console.WriteLine( x.Length );
```

الناتج :

5

مثال :

أبحث عن الحرف m في نص يُدخله المستخدم.

```

static void Main( )
{
    string search = Console.ReadLine( );
    bool check = false;
    for (int i = 0; i < search.Length; i++)
    {
        if ( search[ i ] == 'm' )
            check = true;
    }
    if (check == true)
        Console.WriteLine("exist ");
    else
        Console.WriteLine("not exist");
}

```

في هذا الكود عرفنا متغير نوعه string ثم أخذنا قيمته من المستخدم، وعرفنا متغير نوعه bool يأخذ القيمة true أو false وقد مررت له قيمة ابتدائية false أي أن الحرف الذي نبحث عنه غير موجود إلا أن نجد، ثم أنشأنا حلقة باستخدام for وعداد يبدأ من القيمة صفر إلى أقل من القيمة search.Length والتي تُمثل حجم النص أو عدد حروفه. من الممكن أن تسأل نفسك لماذا لا نكتب عدد حروف النص بشكل مباشر؟! ببساطة لأنه في بعض البرامج لا نعرف عدد حروف النص لأنها تأتي من المُستخدم فلا يُمكن التوقع بحجم النص، ويُفضل دائماً استخدام هذه الخاصية في حالة المصفوفات.

وبداخل الحلقة for نستخدم شرط أنه إذا وُجِدَ الحرف الذي نبحث عنه نجعل قيمة المتغير check = true

ولو لم يجده فتظل قيمته بـ false كما كانت وهذه فائدة أن تمرر قيمة ابتدائية بـ false.

وبعد أن تنتهي الحلقة نستخدم شرط آخر لتأكد هل تغيرت قيمة check أي هل الحرف الذي نبحث عنه موجود أم لا.

مثال :

استقبل نص من المستخدم ثم أعكس هذا النص.

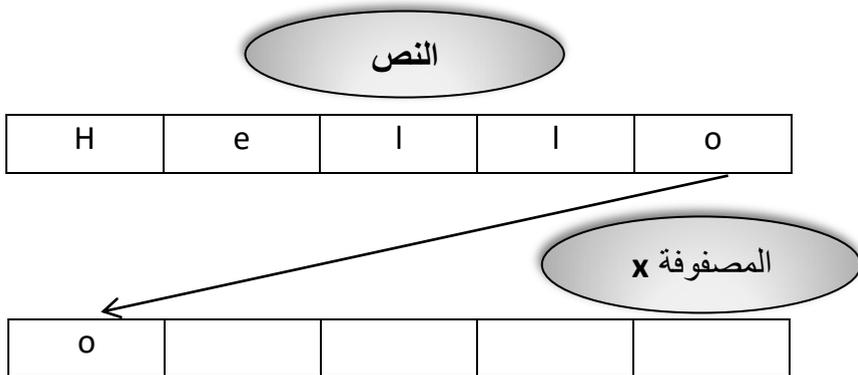
Hello -----> olleH

```
static void Main( )
{
    string str = Console.ReadLine( );
    string str2 = string.Empty; ;
    char[ ] x = new char[ str.Length ];
    for (int i = 0; i < str.Length; i++)
    {
        x[ i ] = str[ str.Length - 1 - i ];
        str2 += x[ i ];
    }
    Console.WriteLine( str2 );
}
```

أخذنا القيمة من المستخدم ثم أنشأنا مصفوفة نوعها char أي من النوع حرف ويكون حجمها نفس حجم النص ولذلك استخدمنا الخاصية str.Length لكي ترجع لنا حجم النص، ثم أنشأنا حلقة وجعلنا كل حرف من نهاية النص يُخزّن في بداية المصفوفة. وأنشأنا متغير آخر نوعه string لكي نُعيد فيه تخزين الحروف التي تأتي من المصفوفة.

ملاحظة :

string.empty هذه لتفريغ المتغير من أي قيم موجودة بهذا المكان مسبقاً.



أو يمكننا الإستغناء عن المصفوفة التي من النوع char لكني تعمدت كتابتها بهذه الطريقة لتعرف طريقة التعامل مع النص والمصفوفة التي من النوع char وأنهما متماثلان.

```
static void Main( )
{
    string str = Console.ReadLine( );
    string str2 = string.Empty; ;
    for (int i = 0; i < str.Length; i++)
        str2 += str[ str.Length - 1 - i ];
    Console.WriteLine( str2 );
}
```

تمارين :

أمامك مصفوفة مكونة من [1, 4, 5, 8, 8, 9, 9, 0] أطلع أكبر عدد مكون من رقمين متجاورين.

في هذا المثال يكون الناتج 99

```
static void Main( )
{
    int[ ] arr = { 1, 4, 5, 8, 8, 9, 9, 0 };
    int max = arr[0];
```

```

string res = string.Empty;
int num = 0;
for (int i = 0; i < arr.Length - 1; i++)
{
    res = arr[ i ].ToString( ) + arr[ i + 1 ].ToString( );
    num = Convert.ToInt32( res );
    if (num > max)
        max = num;
}
Console.WriteLine(max);
}

```

في هذا الكود عرّفنا المتغير max نوعه int لكي يحمل أكبر قيمة موجودة، ثم عرّفنا المتغير res ونوعه string لكي يحمل كل عنصرين متجاورين لأننا سنضطر إلى دمج كل عنصرين متجاورين ولا يمكن الدمج إلا في النصوص لذا سنستقبل كل متغير والذي يليه ثم سنحولهم إلى نصوص من خلال الدالة ToString() وبذلك يتحول الرقمين إلى نص ثم ندمجهم ونخزنهم في المتغير res، بعد ذلك نُحوّل النص المكون من الرقمين المدمجين إلى أعداد صحيحة مرة أخرى لكي نستطيع مقارنتهم بالأعداد الأخرى، لأنه لا يمكن مقارنة نص بعدد صحيح. توجد الكثير من الخصائص والدوال الجاهزة الخاصة بالنصوص.

Length	خاصية تحسب عدد حروف النص
string.Concat()	لدمج أكثر من نص مع بعضهم
Contain()	للبحث عن مقطع معين داخل النص وتُعيد true أو false
EndsWith()	للتأكد من نهاية النص هل ينتهي بمقطع أو حرف معين أنت تحدد أم لا فهي تعيد true أو false
StartsWith()	للتأكد من بداية النص هل يبدأ بمقطع أو حرف معين أنت تحدد أم لا وتعيد true أو false
ToUpper()	لتحويل النص إلى حالة الحروف الكبيرة (capital)

ToLower()	لتحويل النص إلى حالة الحروف الصغيرة (small)
Substring()	لقطع حرف أو أكثر من النص
Replace()	لإستبدال حرف أو مقطع في النص بقيمة جديدة

ويوجد الكثير من الدوال الأخرى الخاصة بالنصوص.
مثال :

```
string x = "C# is a programming language" ;
string y = " and it's a high level language";
Console.WriteLine( x.Length ); -----> 28
Console.WriteLine( string.Concat( x , y ) );
C# is a programming language and it's a high level language
Console.WriteLine(x.EndsWith("uage")); -----> true
Console.WriteLine(x.StratsWith("is")); -----> false
Console.WriteLine(x.ToUpper());
C# IS A PROGRAMMING LANGUAGE

Console.WriteLine(x.ToLower());
c# is a programming language
```

```
Console.WriteLine( x.Substring( 6 , 13 ) );
في هذه الدالة نحدد بداية ونهاية المقطع الذي نريده من النص.
a programming
أو يمكننا تحديد البداية فقط وتكون نقطة النهاية هي نهاية النص.
Console.WriteLine( x.Substring( 6 ) );
a programming language
```

```
Console.WriteLine(x.Replace("pro","PRO"));
C# is a PROgramming language
```

مثال :

أكتب برنامج يحول النص إلى النظام الثنائي (Binary System).

```

using System;
namespace Test
{
    class Program
    {
        static void Main( )
        {
            string str = "Mahmoud soliman";
            string result ;
            int bin = 0;
            Console.WriteLine( str );
            for (int i = 0; i < str.Length; i++)
            {
                result = string.Empty;
                bin = str[ i ];
                Console.Write(str[ i ] + " = " + bin + " ---> ");
                while ( bin != 0 )
                {
                    result = ( bin % 2 ).ToString( ) + result;
                    bin /= 2;
                }
                Console.WriteLine(result + " ");
            }
        }
    }
}

```

في هذه العملية المتغير bin يأخذ القيمة العددية للحرف المحدد من النص.

عملية تحويل الرقم إلى النظام الثنائي من خلال أخذ باقي القسمة له على 2 ثم قسمته على 2

كل حرف يكون له قيمة عددية صحيحة مناظرة له كما ذكرت في مقدمة الكتاب والنص هو عبارة عن مجموعة من الحروف حتى المسافة الفارغة تُعد حرفاً فإننا نأخذ القيمة العددية لكل حرف ونحولها إلى النظام الثنائي ونطبع قيمة أول حرف بالنظام الثنائي ثم نفرغ المتغير الذي نجمع فيه القيم التي تكون بالنظام الثنائي ليكون مُهيئاً لإستقبال قيمة الحرف الذي يليه.

13

البرمجة كائنية التوجه

(Object Oriented Programming)

(OOP)

البرمجة كائنية التوجه هي عبارة عن طريقة لكتابة الكود لكنها جعلت الكود أسهل وأختصرت فيه كثيراً ووفرت من الوقت والمجهود بشكل كبير جداً فهي تتعامل مع كل شيء على أنه كائن وهذا الكائن له صفات ووظائف فمثلاً الإنسان هو كائن له صفات تميزه ووظائف يقوم بها.

فمثلاً من صفاته (لون البشرة : لون العين : الطول :)

ومن أفعاله (يأكل : يمشي : ينام :)

وهكذا....

إذاً فكل كائن من نفس النوع يمتلك نفس اسم الصفات والأفعال لكن بتفاصيل مختلفة، أي تختلف بيانات هذه الصفات والأفعال لأن لكل كائن صفاته وأفعاله الخاصة به.

فالبرمجة كائنية التوجه قامت على هذا الأساس، فمثلاً إذا كنت تريد إنشاء حساب على موقع الفيس بوك، عند إدخال بياناتك هل يكتب لك مصمم الموقع كود برمجي خاص بصفتك عند إنشائك لحساب على الموقع؟ بالطبع لا، عندما تنشئ حساباً لك فإنك تأخذ نسخة من الموقع الأصلي لكن البيانات مجردة أي بدون بيانات وتملاًها أنت ببياناتك الخاصة،

في ذلك الوقت أنت أخذت كائن (نسخة) من الموقع وأدخلت فيه بياناتك أنت، وشكل الموقع ووظائفه متشابهة تماماً عند كل الناس والبيانات المجردة أيضاً متشابهة عند كل الناس فمثلاً هو يطلب منك تحديد الجنس والعمر وهكذا وأنت تدخل بياناتك، هذه هي البيانات المجردة أي أنها لا تزال لا تحتوي على قيم، وأي شخص جديد يريد إنشاء حساب فإنه سيأخذ نفس النسخة من الموقع ويملأها ببياناته الخاصة.

هذه مجرد أمثلة تشبيهية فقط.

وهذه هي الطريقة التي بنيت عليها لغة السي شارب كما قولنا في مقدمة الكتاب.

إذاً أين يكتب الكود الذي يأخذ منه الكائن نسخة من البيانات التي بداخله؟

يكتب بداخل class والـ class هو البنية التحتية للبرمجة كائنية التوجه.

وتكمن قوة الـ class في عملية تسمى عملية تغليف البيانات، أي يحدد من يستطيع الوصول إلى البيانات والتعامل معها.

وستحدث عن كل شيء يخص البرمجة كائنية التوجه والـ class. لكن أولاً سنتحدث عن طرق الوصول للبيانات أو التغليف.

14

التغليف

(Encapsulation)

توجد ٥ أنواع لتغليف البيانات في لغة السي شارب، أو تسمى طرق الوصول للبيانات. قبل البداية في شرح هذه الأنواع سأذكركم أن المشروع أو البرنامج هو عبارة عن مجموعة من namespaces والـ namespaces عبارة عن مجموعة من الـ classes.

يمكننا إنشاء أكثر من namespace في المشروع ويمكننا إنشاء أكثر من class بداخل نفس الـ namespace، ويكون بداخل كل class الخصائص والدوال (الصفات والوظائف) الخاصة به.

في هذه الحالة نحتاج إلى حماية بعض البيانات التي تكون بداخل الـ class مثلاً من يستطيع رؤيتها والتعامل معها بداخل المشروع، فمن الممكن أن تكون بعض البيانات لـ class مرئية في class آخر وبعض البيانات غير مرئية ولا يمكن التعامل معها إلا بداخل الـ class الذي كُتِبَتْ بداخله فقط، المبرمج يحدد من يستطيع رؤية البيانات بداخل المشروع سواء class في نفس الـ namespace أو class في namespace أخرى بداخل المشروع.

أول طريقة للتغليف هي :

public - 1

البيانات التي تكون من هذا النوع فإنها ستكون مرئية في كل المشروع ويستطيع أي class داخل namespace أو أي namespace أخرى رؤيتها والتعامل معها.

طريقة تعريف الدوال أو الخصائص تكون بهذا الشكل :

طريقة الوصول + نوع البيانات + الأسم

مثال :

public int variable;

```
using System;
namespace First_program
{
    class Program
    {
        static void Main( )
        {
            Program2 pro = new Program2( );
            pro.x = 10;
        }
    }
    class Program2
    {
        public int x;
    }
}
```

كل ما عليك معرفته في هذا الكود أننا استطعنا إسناد قيمة للمتغير X في الـ class الآخر Program، وأستطعنا رؤيته والتعامل معه. ويتم الوصول إلى محتوى الـ class من خلال كائن أو نسخة (object) من هذا الـ class ثم النقطة (.) (dot)

النوع الثاني وهو :

private – 2

هذا النوع يجعل إستخدام البيانات التي من نفس نوعه تقتصر على الـ class المكتوبة بداخله فقط، أي أنه يحرم إستخدام البيانات التي من نوعه إلى أي class آخر حتى لو class في نفس الـ namespace ويمكننا التعامل مع البيانات التي من هذا النوع بداخل الـ class الذي كُتبت به فقط، ولو حاولنا تنفيذ الكود السابق على النوع private فإن المترجم سيخرج لك خطأ أنه لا يمكن الوصول لهذا المتغير بسبب الحماية التي يمتلكها.

```

namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            pro p = new pro();
            p.x = 10;
        }
    }
}

class pro
{
    private int x;
}

```

int pro.x
Error: 'ConsoleApplication3.pro.x' is inaccessible due to its protection level

النوع الثالث :

protected – 3

وهذا النوع معناه أنه لا يمكننا رؤية البيانات ولا التعامل معها في أي class آخر إلا إذا وُرثَ الـ class الذي كُتِبَتْ بداخله البيانات التي من هذا النوع. وسنفهم مفهوم الوراثة لاحقاً.

```
using System;
namespace First_program
{
    class Program : Prog
    {
        static void Main( )
        {
            Program p = new Program( ) ;
            p.x = 10 ;
        }
    }
    class Prog
    {
        protected int x ;
    }
}
```

النقطتان تعني أن الـ class " Program " قد ورثَ الـ class " prog " ويمكنه التعامل مع البيانات التي بداخله كأنها كُتِبَتْ بداخله لكن ليس كل البيانات، فقط المسموح به، مثل public و protected، إلى الآن، وعندما نصل إلى الوراثة سنفهم كل شيء عنها.

Program p=new Program(); <-- هذه الخطوة

تعني أننا أخذنا نسخة (كائن) من هذا الـ class (Program)

كائن (object) <----- p

class <----- Program

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApplication1
8  {
9      class Program : pro
10     {
11         static void Main(string[] args)
12         {
13             Program p=new Program();
14             p.x = 10;
15         }
16     }
17     class pro
18     {
19         protected int x;
20     }
21 }
22

```

النوع الرابع :

internal – 4

وهي تعني أننا نستطيع رؤية البيانات التي من هذا النوع في كل الـ assembly للبرنامج

الـ assembly يمكن أن تحتوي على أكثر من namespace

exe أو الـ DLL <---assembly ملف الـ

النوع الخامس :

protected internal – 5

نستطيع رؤية البيانات والتعامل معها في كل الـ assembly لكن يجب أن يكون الـ class الذي كُتِبَتْ بداخله موروث للـ class الذي سنستخدمها فيه.

protected internal int x;

15

(Static)

المتغيرات والدوال التي بداخل الـ class ويكون نوعها static تكون ثابتة أي أنه لا يمكن التعامل معها إلا من خلال اسم الـ class نفسه.

نستخدم static عندما نريد التعامل مع الدالة أو الحقل خارج الـ class بدون الحاجة إلى أخذ نسخة منه (object) فمثلاً الدالة (WriteLine) هي دالة static داخل Console والـ Console هو الـ class الخاص بدوال الإدخال والإخراج. فعند إستدعاء هذه الدوال نستخدم اسم الـ class مباشرةً بدون أن نأخذ منه نسخة.

والدالة الرئيسية يجب أن تكون من النوع static، وأي دالة نريد إستدعائها مباشرة في دالة من النوع static يجب أن تكون static. بإختصار أي نوع static لا نستطيع التعامل معه مباشرة إلا من خلال نوع static آخر، لكن يمكن التعامل مع المتغيرات التي من نوع static من خلال الدوال، حتى لو لم تكن الدالة static لكن العكس خطأ، أي أننا لا نستطيع إستخدام متغير ليس static مع دالة نوعها static.

مثال :

```
using System;
namespace First_program
{
    class Program
    {
        int x;
        static void Main( )
        {
            x = 10;
```

```

        خطأ لأن الدالة من النوع static
        وطالما الدالة static يجب أن يكون المتغير static أيضاً
    }
}

```

```

using System;
namespace First_program
{
    class Program
    {
        static int x;
        static void Main( )
        {
            x = 10;
        }
    }
}

```

في هذه الحالة يجوز إسناد قيمة للمتغير والتعامل معه بشكل طبيعي لأن نوعه static والدالة نوعها static.

```

using System;
namespace First_program
{
    class Program
    {
        static int x;
        void name( )
        {
            x = 4;

```

```
// يجوز إسناد قيمة للمتغير الذي من النوع static من خلال الدالة في الحالة العادية
    }
    static void Main( )
    {
        }
    }
}
```

مثال ٢ :

```
using System;
namespace First_program
{
    class A
    {
        public static int x ;
    }
    class B
    {
        public static void variables()
        {
            A.x = 5 ;
        }
    }
}
```

16

الدوال

(Methods or Functions)

تحدثنا في مقدمة الكتاب عن الدوال أو الوظائف وما عملها، قبل وجود فكرة الدوال كنا إذا أردنا استخدام جزء معين من الكود الذي كتبناه مرة أخرى فكان يجب كتابته ثانيةً كلما أحتجنا إليه، فكان يأخذ وقت ومجهود، فلما لا نكتب جزء معين من الكود مرة واحدة ونستخدمه متى أحتجنا إليه؟!

إذاً ما الكود الذي يُكتب بداخل الدالة؟!

نكتب الكود الذي نريد استخدامه أكثر من مرة، أي كود نريد تنفيذه. والدوال تُستخدم لإعطاء قيم لخواص الـ class والتعامل معها. وُجِدَتْ فكرة الدوال قبل وجود البرمجة كائنية التوجه، والدوال عامل رئيسي في البرمجة فلا يمكن الإستغناء عنها.

أي دالة يكون لها قيمة عائدة لما كُتِبَ بداخلها، ما عدا نوع واحد من الدوال (void) لا يعيد قيمة، معنى قيمة عائدة أن الدالة كلها يكون لها قيمة واحدة يمكن استخدامها وإدخالها في عمليات أخرى، سنفهم معنى القيم العائدة بمثال. أولاً سنعرف طريقة تعريف الدالة.

يمكننا تسمية الدالة بأي اسم نريده.

() + اسم الدالة + القيمة العائدة من الدالة + طريقة الوصول

{

}

Access_Modifier + Return_Type + Name_of_function + ()

```
{
    // نكتب الكود داخل القوسين هنا .
}
```

طريقة الوصول تعني صلاحيات استخدام الدالة، أين نستخدمها ومن يستطيع استخدامها.

قد تكون :

public , private , protected , internal , protected internal.

اسم الدالة :

توجد طرق لكتابة أسماء أعضاء الـ class سواء كان متغير أو دالة، هي مجرد طرق متعارف عليها، يمكننا التغاضي عن هذه الطرق وكتابتها بالشكل الذي نريد، لكن يُفضل كتابتها بهذه الطرق.

أول طريقة :

Pascal case - 1

يكون أول حرف من الاسم حرف كبير (Capital) وإذا كانت الكلمة مكونة من مقطعين فيكون أول حرف من المقطع الأول كبير وأول حرف من المقطع الثاني كبير مثل :

UserName

أو User_Name

ثاني طريقة :

Camel case – 2

وفي هذه الحالة يكون أول حرف من المقطع الأول صغير و أول حرف من المقطع الثاني كبير مثل :

userName

passWord

وهكذا...

ثالث طريقة :

UpperCase – 3

هذه الحالة تكون كل حروف الاسم كبيرة وتستخدم مع الاسماء التي تمتلك حروف قليلة مثل حرفين أو ثلاث ومع الثوابت.

مثال :

```
const string NAME = "mahmoud" ;
```

```
int GPA;
```

وهكذا ...

يجب أن يكون اسم الدالة متصل ولا تفصله مسافة فمثلاً لا نستطيع تسمية دالة أو متغير

بهذا الشكل user name

لانه توجد مسافة تفصل الاسم وهذا لا يجوز، وفي حالة الأسماء التي تتكون من أكثر

من مقطع مثل First Name ... نستخدم (_) فتكون First_Name.

أنتهينا من تسمية الدالة.

نوع الدالة يكون :

int , float , double , long , short , string , char , bool

هذه الأنواع تعني أن نوع الدالة يكون من نفس نوع القيمة العائدة منها فمثلاً إذا كانت الدالة int إذا يجب أن تعيد قيمة صحيحة ولو كانت string يجب أن تعيد نص أو كلمة، float يجب أن تعيد قيمة كسرية. bool تعيد true أو false وهكذا...

سنفهم معنى القيمة العائدة للدالة لكن مؤقتاً حاول أن تتعلم كيفية بناء هيكل الدالة وماذا يُكْتَب بداخلها وكيف تُسْتَحْدَم.

بعد كتابة الدالة والإنتهاء منها كيف نستخدمها وكم مرة نستخدمها؟

عن طريق إستدعاءها بإسمها فقط ونستدعيها بعدد المرات التي نريد، نستطيع إستدعاءها وإستخدامها ملايين المرات بدون إعادة كتابة الكود المكتوب بداخلها.

اتفقنا أن الدالة يكون لها قيمة واحدة عائدة ما عدا نوع واحد فما هو؟

هذا النوع هو void أي لا ترجع أي شيء ولا يكون لها قيمة نستطيع إستخدامها في عمليات أخرى، أي عندما ننتهي من تنفيذ الدالة لا نريد منها أي شيء آخر.

ملاحظة :

سنستخدم الـ class الرئيسي وسنكتب بداخله الدوال التي نريدها. سنُطَبِّق عليه "مؤقتاً". ويكون بداخله الدالة الرئيسية التي يبدأ تنفيذ الكود منها. يمكننا كتابة الدالة الجديدة فوق الدالة الرئيسية (Main) أو تحتها، لا يوجد فرق.

" نُكْتَب الدوال بداخل class "

مثال :

```
using System;
namespace Anything
{
    class Program
    {
```

```

    سنكتب كل الدوال هنا
    static void Main( )
    {
        }
    }
}

```

هذا هو الشكل الذي سنعمل عليه .

مثال:

أكتب دالة تطبع Hello World.

قبل البدء في كتابة الكود يجب أن نحلل السؤال أولاً، بعدها نكتب الخوارزميات لهذه المسألة " الخوارزميات أي نحدد خطوات الحل خطوة بخطوة ونحل المسألة بطريقة جبرية "، إذا سرنا على هذا النمط فلن نواجه أي صعوبة في كتابة أي كود.

لنحلل هذه المسألة ...

نريد إنشاء دالة تطبع Hello World هل نحتاج قيمة لهذه الدالة فيما بعد؟ هل سنجري عليها أي عملية؟ جمع أو طرح أو قسمة أو سنأخذ قيمتها ندخلها في دالة أو متغير آخر؟ إذا كان الجواب لا ... فلن نحتاج أن يكون لها قيمة عائدة وستكون نوع الدالة void. ونسميها بالاسم الذي نريد لكن يفضل أن يكون اسم معبر عن الوظيفة التي تقوم بها الدالة، مثلاً إذا كانت دالة طباعة.. نسميها مثلاً print أو show أو view أي اسم نريده فلن يحدث هذا فارق في أي شيء.

ملاحظة :

طريقة الوصول لن تصنع فارق هنا، لأننا لن نستخدم الدالة في class خارجي إنما سنستخدمها بداخل الـ class المكتوبة بداخله فلن يفرق ما إذا كانت public أو private أو protected الكل سيان في هذا الوقت. اتفقنا أن هذا لمجرد التطبيق مؤقتاً فقط وبعدما ننتهي من تعلم كتابة الدوال والتعامل معها سنتعلم كيف نستخدمها في أكثر من class آخر.

```

using System;
namespace ConsoleApplication1
{
class Program
{
    public void print()
    {
        Console.WriteLine("Hello World") ;
    }
}
}

```

إذا أنتهينا من كتابة الكود الخاص بالدالة كيف سننفذها وأين؟

عندما تفتح visual studio فأنت تجد class مكتوب والدالة الرئيسية التي يبدأ تنفيذ الكود منها مكتوبة بداخله.

فنستدعي الدالة التي أنشأناها بواسطة كتابة اسمها في الدالة الرئيسية وبما أن الدالة الرئيسية نوعها static فيجب أن تكون الدالة التي سنكتبها نوعها static أيضاً حتى تستطيع الدالة الرئيسية رؤيتها، يمكننا إستدعائها بطريقة أخرى عن طريق إنشاء كائن من هذا الـ class لكن عندما نصل إلى شرح الـ class وكيفية التعامل معه وطريقة إنشاءه سنتعلم كل شيء يخص الدوال وكل طرق التعامل معها بداخل الـ class.

حالياً أي دالة نوعها static يجب أن يكون ما سنستدعيه بداخلها سواء دالة أخرى أو متغير من نفس نوعها.. أي static.

وسيتم تعريفها بهذا الشكل :

طريقة الوصول + static + القيمة العائدة من الدالة + اسم الدالة () { الكود }

Access_Modifier + static + Return_Type + Name_of_function + ()

{ The code }

مثال :

```

using System;
namespace ConsoleApplication1
{
class Program
{
    public static void print()
    {
        Console.WriteLine("Hello World");
    }
    static void Main( )
    {
        print ( ) ;
    }
}
}

```

عند كتابة اسم الدالة ونُشغل البرنامج سيذهب إلى المكان الذي كتبنا فيه كود الدالة ويتم تنفيذ ما بداخلها.

سيطبع Hello World

في عملية إنشاء الدالة إذا كنا سنستخدم متغيرات في هذه الدالة فيمكن أن نمرر هذه المتغيرات للدالة أثناء التعريف وتسمى في هذا الوقت (Parameters) لا يوجد حد معين لعدد مدخلات دالة " parameters " فقط ما سنستخدمه وما نحتاج إليه، وهذه المدخلات عند استدعاء الدالة سنضطر إلى تمرير قيمها إلى الدالة حتى تحدث العملية التي نريدها من الدالة، وإعطاء مُدخَل للدالة هو أمر اختياري لكن هناك بعض الدوال يُفضل تمرير مُدخلات لها لإجراء عمليات عليها وتمييز كل دالة عن الأخرى.

فمثلاً عند تسجيل الدخول لحساب فيس بوك فإنك تدخل البريد الإلكتروني وكلمة المرور، فهذه البيانات التي تدخلها تكون مُدخل لدالة تفحص كلمة المرور والبريد الإلكتروني الخاصين بك المُسجلين على قواعد البيانات.

ويتم تعريف المُدخل بالطريقة العادية لانه متغير عادي.

Access_Modifier + static + Return_Type + Name_of_function +
(parameter1,parameter2,.....)

مثال :

انشيء دالة لجمع رقمين ثم اطبع ناتج الجمع.

```
using System;
namespace ConsoleApplication1
{
class Program
{
    public static void Print_Sum( int num1 , int num2 )
    {
        int sum = num1 + num2 ;
        Console.WriteLine( sum ) ;
    }
static void Main( )
{
    int number1 , number2 ;
    number1 = int.Parse(Console.ReadLine() ) ;
    number2 = int.Parse(Console.ReadLine() ) ;
    print( number1 , number2 ) ;
}
}
}
```

في هذا الكود أنشأنا الدالة وأنشأنا لها مُدخلين وتمت عملية الجمع بداخل الدالة، ثم أستدعينا دالة جمع الرقمين في الدالة الرئيسية، لكن الدالة التي أنشأناها تأخذ مُدخلين لذا سنضطر إلى تعريف متغيرين وتمريرهما إليها وقد أخذنا قيم المتغيرات من المستخدم، عند تشغيل البرنامج وعندما يصل المترجم إلى السطر الذي استدعينا به الدالة يذهب إلى هذه الدالة أيًا كان موقعها في الكود وينفذ ما بداخلها.

الدالة التي من النوع void لا نستطيع عمل أي عمليات عليها فلا يمكن أن ندخلها في أي عملية أخرى ولا نستطيع إجراء عملية طباعة عليها باستخدام دالة `Console.WriteLine()`

ولا نستطيع فعل ذلك :

```
int temp = print( number1 , number2 ) +2;
```

لا نستطيع تمريرها إلى متغير أو فعل أي شيء فيها لأنها ليس لها قيمة عائدة.

ماذا إذا أردنا أن نستخدم قيمة ناتج الجمع ونسندها إلى متغير آخر؟

لنحلل المسألة ...

ناتج جمع رقمين من النوع int ماذا سيكون نوع ناتج الجمع؟

عدد صحيح أيضاً.

إذا سيكون نوع القيمة العائدة من هذه الدالة int فيكون نوع الدالة int.

ويجب في هذه الحالة أن نستخدم الكلمة المحجوزة في اللغة return في نهاية الكود نكتب القيمة العائدة من هذه الدالة، وكل دالة يكون لها قيمة واحدة فقط عائدة.

القيمة + return

مثال :

```
using System;
namespace ConsoleApplication1
{
```

```

class Program
{
    public static int Print_Sum( int num1 , int num2 )
    {
        int sum = num1 + num2 ;
        return sum ;
    }
    static void Main( )
    {
        int number1,number2 ;

        number1 = int.Parse(Console.ReadLine( )) ;

        number2 = int.Parse(Console.ReadLine( )) ;

        int temp = print( number1 , number2 ) + 100 ;

        Console.WriteLine( temp ) ;

    }
}
}

```

في هذا الكود أنشأنا الدالة وأصبح نوعها `int` أي القيمة العائدة منها `int` الدالة كاملة في ذلك الوقت هي عبارة عن قيمة واحدة يمكن أن نعمل بها أي شيء، وأستدعيها في الدالة الرئيسية ومررنا لها المُدخلات التي عرفناها وأخذنا قيمهم من المستخدم، بعد ذلك مررنا الدالة كاملة إلى متغير، إذا هذا المتغير حالياً يحمل قيمة الدالة. ثم أضفنا إلى قيمة الدالة ١٠٠، فمثلاً إذا أدخل المستخدم ٢٠ و ٣٠ سيكون ناتج الدالة ٥٠ وسيضيف عليها ١٠٠ ثم سيضع القيمة الجديدة في المتغير `temp`.

ماذا لو كانت العملية التي ستتم في الدالة هي قسمة وليست جمع إذا فيوجد احتمال أن يكون ناتج القسمة عدد كسري، فيجب أن نغير نوع الدالة لنفس نوع القيمة العائدة منها. أي يكون نوعها `float` أو `double`.

ملاحظة :

مُدخلات الدالة أثناء الإنشاء تسمى " Parameters " أما أثناء إستدعاء الدالة وتمرير قيم لها فتُسمى المُدخلات " Arguments " .

مثال :

أكتب دالة تأخذ رقمين ثم تعيد الرقم الأكبر فيهما .

```
using System;
namespace ConsoleApplication1
{
class Program
{
    public static int Max_Value( int num1 , int num2 )
    {
        if ( num1 > num2 )
            return num1;
        else if ( num2 > num1 )
            return num2 ;
        else
            return 0 ;
    }
    static void Main( )
    {
        int number1 , number2;
        number1 = int.Parse(Console.ReadLine( ));
        number2 = int.Parse(Console.ReadLine( ));
        Console.WriteLine(Max_Value( number1 , number2 ) ;
    }
}
```

```

    }
}
}

```

تمارين :

أكتب دالة تأخذ من المستخدم الأس والأساس وتخرج الناتج.

مثلاً الاساس ٢ والأس ٥ = ٣٢

الأساس ٣ والأس ٤ = ٨١

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Power( int Base_num , int power_num )
        {
            int result = 1;
            for ( int i = power_num ; i > 0 ; i-- )
            {
                result *= Base_num ;
            }
            Console.WriteLine( result );
        }
        static void Main( )
        {
            int Base = int.Parse(Console.ReadLine( ) );
            int Power = int.Parse(Console.ReadLine( ) );

```

```

        Power ( Base,Power ); // ( 2,6 )
    }
}
}

```

الناتج :

64

وتوجد دالة جاهزة بداخل اللغة تسمى pow داخل class يسمى Math

```
Math.Pow ( 2 , 6 );
```

سيخرج لك نفس النتيجة لكن يجب أن تعتمد على نفسك في مرحلة التعلم وتتعلم كيف تنشئ دوال بنفسك .

تمارين :

أكتب دالة تأخذ رقم من المستخدم وتعيد الجذر التربيعي لهذا الرقم إن وُجد.

```

using System;

namespace ConsoleApplication1
{
    class Program
    {
        public static void SQRT( int num )
        {

```

```

        for ( int i = num ; i > 0 ; i-- )
        {
            if ( i * i == num )
            {
                Console.WriteLine ( i );
            }
        }
        Console.WriteLine ( result );
    }
    static void Main( )
    {
        int number=int.Parse ( Console.ReadLine ( ) );
        SQRT ( number ); //( 25 )
    }
}

```

الناتج :

5

تمارين :

أكتب دالة تأخذ من المستخدم مصفوفة ورقم معين ثم ترجع عدد مرات تكرار هذا الرقم في تلك المصفوفة.

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static int count_item( int [ ]arr, int num )
        {

```



```
{
    static float Math_Operation(float operand1, char operation,
                                float operand2)
    {
        float result = 0.0f;
        switch ( operation )
        {
            case '+':
                result = operand1 + operand2;
                break;
            case '-':
                result = operand1 - operand2;
                break;
            case '/':
                result = operand1 / operand2;
                break;
            case '*':
                result = operand1 * operand2;
                break;
            case '%':
                result = operand1 % operand2;
                break;
        }
        return result;
    }
}
static void Main( )
{
    while ( true )
    {
        Console.WriteLine("Enter operand1 : operation : operand2 :");
```

```

float op1 = float.Parse(Console.ReadLine( ));
char op = char.Parse(Console.ReadLine( ));
float op2 = float.Parse(Console.ReadLine( ));
Console.WriteLine(Math_Operation(op1 , op , op2));
}
}
}
}

```

الدوال التي تعيد قيم يمكن تمرير القيمة العائدة منها إلى دالة أخرى .

مثال :

```

using System;
namespace ConsoleApplication6
{
class Program
{
static int a(int x , int y)
{
if (x > y)
return x;
else
return y;
}
static int b(int x, int y)
{
return x + y;
}
static int c(int x, int y)
{
return x - y;
}
static int d(int x, int y)
{
return x * y;
}
}
}

```

```

}
static void sum(int a,int b,int c)
{
    Console.WriteLine(a + b + c);
}
static void Main( )
{
    sum(a(7, b(3, c(5, d(2,4) ) ) ), 5, 2);
}
}
}

```

الناتج : ١٤

في هذا الكود أنشأنا ٥ دوال، كل دالة لها وظيفة مختلفة عن الأخرى، فالدالة "a" تأخذ قيمتين ثم تعيد القيمة الأكبر فيهما والدالة "b" تأخذ قيمتين ثم تعيد ناتج جمعهما، والدالة "c" تأخذ قيمتين وتعيد ناتج طرحهما والدالة "d" تأخذ قيمتين وتعيد ناتج ضربهما، والدالة "sum" تأخذ ثلاث قيم وتطبع ناتج جمعهم.

وفي دالة الـ Main أستدعينا الدالة sum وهي تأخذ ثلاث قيم كما ذكرت، ثم مررنا لها الدالة a وبعد ذلك مررنا للدالة a القيمة ٧ والقيمة الأخرى كانت عبارة عن الدالة b ومررنا للدالة b القيمة ٣ والقيمة الأخرى كانت عبارة عن الدالة c ومررنا لها القيمة ٥ والقيمة الأخرى كانت عبارة عن الدالة d ومررنا للدالة d القيمتين ٢ و ٤.

عند تنفيذ هذا الكود فإن الدوال الداخلية هي التي تُنفَّذ أولاً وبما أن هذه الدوال ذات قيمة عائدة فإن كل دالة تُعيد قيمة واحدة فقط. وتكون هذه القيمة هي المُدخل الآخر للدالة التي تسبقها .

في هذا المثال الدالة d تأخذ القيمتين ٢ و ٤ وتعيد ناتج ضربهما أي ٨ إذاً تكون القيمتان اللتان مررناهما للدالة c هما ٥ و ٨ ، ثم تعيد الدالة c ناتج طرحهما فتعيد القيمة -٣ ، وبذلك تكون القيم الممررة إلى الدالة b هي ٣ و -٣ ، ثم تعيد الدالة b ناتج جمعهما فتعيد القيمة صفر، وبذلك تكون القيم الممررة للدالة a هي صفر و ٧ وتعيد الدالة a القيمة الأكبر فيهما فتعيد القيمة ٧ ،

وبذلك تكون القيم التي مُررت للدالة sum هي : (7 , 5 ,2)

وتطبع ناتج جمعهم فيكون ١٤

ref

عندما نمرر قيم للدالة فإننا نأخذ نسخة من هذه القيم ويحدث التغيير فيها أما القيم الأساسية في ذاكرة الكمبيوتر لم يحدث بها أي تغيير ولم تتأثر.

فمثلاً إذا أردنا تنفيذ عملية تبديل بين قيمتي متغيرين عن طريق دالة فإنه سيبدل القيم لكن سيبدلها في النسخة فقط وليس في المتغيرات الأصلية.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Swap( int num1 , int num2 )
        {
            int temp=num1;
            num1=num2;
            num2=temp;
        }
        static void Main( )
        {
            int number1 = 10 , number2 = 20;
            Swap ( number1 , number2 ) ;
            Console.WriteLine("number1 = " + number1);
            Console.WriteLine("number2 = "+number2);
        }
    }
}
```

الناتج :

```
number1 = 10;
```

```
number2 = 20;
```

لن يحدث أي تغيير فعلي في القيم لأننا غيرنا في النسخ فقط. لكن ماذا لو أردت إحداث تغيير فعلي في المتغيرات الأصلية في الذاكرة ؟

نستخدم الكلمة المحجوزة (**ref**)

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Swap( ref int num1 ,ref int num2 )
        {
            int temp = num1;
            num1 = num2;
            num2 = temp;
        }
        static void Main( )
        {
            int number1 = 10 , number2 = 20;
            Swap ( ref number1 , ref number2 ) ;
            Console.WriteLine("number1 = " + number1);
            Console.WriteLine("number2 = "+number2);
        }
    }
}
```

الناتج :

number1 = 20;

number2 = 10;

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Swap( ref int num1 ,ref int num2 )
        {
            num1 = 40 ;
            num2 = 30 ;
        }
        static void Main( )
        {
            int number1=10,number2=20;
            Swap ( ref number1 , ref number2 ) ;
            Console.WriteLine("number1 = " + number1);
            Console.WriteLine("number2 = "+number2);
        }
    }
}
```

الناتج :

number1 = 40;

number2 = 30;

القيم الجديدة لن تؤثر في المتغيرات لأننا مررنا للمتغيرات قيم مرجعية في الدالة والقيم وُضعت في المتغيرات الأصلية لا النسخ.

تمارين :

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static int Swap( ref int num1 )
        {
            num1 = num1*num1 ;
            return num1;
        }
        static void Main( )
        {
            int number1 = 10;
            Swap ( ref number1 ) ;
            Console.WriteLine("number1 = " + number1);
        }
    }
}
```

النتاج :

number1 = 100

out

إذا أنشأنا دالة تأخذ مُدخلين ووأردنا إستدعاء هذه الدالة في دالة أخرى أو أي مكان في الكود، ومررنا متغيرات للدالة في الاستدعاء لكننا لم نعطي قيم لهذه المتغيرات ونريد أن نأتي بالقيم من الدالة نفسها، في الحالة العادية هذا لا يجوز ويجب أن نعطي قيم للمتغيرات التي سنمررها كمدخلات للدالة حتي إذا أدخلنا لهذه المتغيرات قيم بتدائية في الدالة.

```

namespace ConsoleApplication3
{
    class Program
    {
        public static void fun(int num1,int num2)
        {
            num1 = 200;
            num2 = 100;
        }
        static void Main(string[] args)
        {
            int number1, number2;
            fun(number1, number2);
            Console.WriteLine(" number1 = " + number1);
            Console.WriteLine(" number2 = " + number2);
        }
    }
}

```

لإستخدام القيم التي أستخدمناها في الدالة يجب أن نحدد له أنه سيأتي بقيم لمُدخلات الدالة من الدالة نفسها.

مثال :

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void fun_name( out int num1 ,out int num2 )
        {
            num1 = 200 ;
            num2 = 100 ;
        }
        static void Main( )
        {
            int number1 , number2;
            fun_name( out number1 , out number2 ) ;
            Console.WriteLine("number1 = " + number1);
            Console.WriteLine("number2 = "+number2);
        }
    }
}
```

الناتج :

number1 = 200

number2 = 100

حتى لو أدخلنا قيم لـ number1 , number2 فإنه سيتجاهلها ويستخدم القيم الخارجية التي تأتي من الدالة.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void fun_name( out int num1 ,out int num2 )
        {
            num1 = 200 ;
            num2 = 100 ;
        }
        static void Main( )
        {
            int number1 = 60 , number2 = 30 ;
            fun_name ( out number1 , out number2 ) ;
            Console.WriteLine("number1 = " + number1);
            Console.WriteLine("number2 = "+number2);
        }
    }
}
```

النتج :

number1 = 200

number2 = 100

ref تتطلب أن تدخل قيم للمتغيرات التي ستمررها للدالة لحظة الإستدعاء.

```

namespace ConsoleApplication3
{
    class Program
    {
        public static void fun(ref int num1,ref int num2)
        {
            num1 = 200;
            num2 = 100;
        }
        static void Main(string[] args)
        {
            int number1, number2;
            fun(ref number1, ref number2);
            Console.WriteLine(" number1 = " + number1);
            Console.WriteLine(" number2 = " + number2);
        }
    }
}

```

الخط الذي تحت المتغيرين لحظة إسنادهما للدالة يُعني أنه يجب أن نعطي لهما قيم وقت تعريفهما.

مثال :

أكتب دالة لتنفيذ عملية المضروب لرقم يدخله المستخدم.

مثال لمضروب الـ ٤ :

$$24 = 1 * 2 * 3 * 4$$

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void factorial ( int num )
        {
            int fact=1;
            if( num<=0 )
                Console.WriteLine( 0 );
            else
            {
                for(int i=1 ; i<=num ; i++)
                {
                    fact *= i ;
                }
                Console.WriteLine(fact);
            }
        }
        static void Main( )
        {
            int number = int.Parse( Console.ReadLine( ));
            factorial(number) ;
        }
    }
}
```

params

تستخدم كلمة params عندما لا نعلم عدد مُدخلات (parameters) الدالة وعدد المُدخلات في هذه الدالة يُعبر عن حجم المصفوفة.

مثال :

```
using System;
namespace ConsoleApplication1
{
class Program
{
    public static int Max_Value_in_array ( params int [ ] arr )
    {
        int max=arr[0] ;
        for( int i =0 ; i < arr.Length ; i++)
            if ( arr [ i ] > max )
                max = arr[ i ];
        return max ;
    }
    static void Main( )
    {
        Console.WriteLine ( Max_Value_in_array ( 3 , 4 , 5 , 6 , 7
            , 8 , 1 , 2 , 3 , 4 , 5 ) );
    }
}
}
```

النتج :

٨ عدد مُدخلات الدالة يكون غير محدود أي يمكننا تمرير العدد الذي نريده لكن مع كل مُدخل نضيفه يزيد حجم المصفوفة بمقدار ١ ، فمثلاً حجم المصفوفة في هذا الكود سيكون ١١ ، وسيطبع القيمة الأكبر في هذه المصفوفة.

17

الدالة ذاتية الإستدعاء

(Recursion function)

يوجد نوع من الدوال يسمى الدالة العائدة، أي التي تُستدعى بداخل نفسها إذا أُستدعت دالة بداخل نفسها فإنها تظل تستدعي نفسها إلى أن توقفها بشرط، لذا فيجب تحديد شرط لها لكي تتوقف.
سننفذ نفس برنامج المضروب بها، بدون إستخدام loop

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static int factorial ( int num )
        {
            if( num<=0 )
                return 0 ;
            else if ( num==1 ) // لتوقف الدالة عند هذا الرقم
                return 1 ;
            else
                return ( num * factorial(num-1));
        }
        static void Main( )
        {
```

```

int number = int.Parse( Console.ReadLine ( ) );
Console.WriteLine ( factorial ( number ) );
}
}
}

```

النتائج :

١٢٠

يجب أن يكون نوع الدالة أي نوع غير void فلا يُمكن أن يكون void، لأن طريقة عمل هذا النوع من الدوال هو أنها تستدعي نفسها مرة أخرى بداخل نفسها فإذا كان نوعها void فإننا لن نستطيع عمل أي عملية عليها فنحن هنا نحتاج إلى أن نضرب الرقم في قيمة الدالة ونطرح ١ من الرقم إلى أن يصل الرقم المُدخل إلى ١.

فإذا أدخل المستخدم صفر أو قيمة أقل من الصفر بالسالب فإنها سترجع صفر،

وإذا أدخل ١ فإن الدالة سترجع ١

وإذا أدخل رقم ٥ فإنها ستضرب الرقم ف قيمة الدالة ناقص ١ وتستدعي نفسها مرة أخرى فيكون الرقم ٤ ثم ٣ ثم ٢ وتتوقف وتعيد ناتج ضرب هذه القيم مع بعضها فتكون قيمة الدالة كلها تساوي ٥ * ٤ * ٣ * ٢ أي ١٢٠.

مثال :

أحسب مجموع الأرقام من رقم يدخله المستخدم إلى رقم أكبر يحدده المستخدم أيضاً بدون استخدام loop.

مثلاً من ٤٠ إلى ١٢٠

```
using System;
```

```
namespace ConsoleApplication1
```

```

{
class Program
{
    public static int sum ( int start , int end)
    {
        if( start>end ) // لتوقف الدالة عند هذا الرقم
            return 0 ;
        else
            return ( start + sum(start+1 , end));
    }
    static void Main( )
    {
        int start = int.Parse( Console.ReadLine( ));
        int end = int.Parse( Console.ReadLine( ));
        Console.WriteLine ( sum ( start , end ) );
    }
}
}

```

مثال :

أنشئ دالة تأخذ مصفوفة كمدخل وتخرج أكبر قيمة بهذه المصفوفة بدون استخدام حلقات تكرار.

```

using System;
namespace ConsoleApplication19
{
    class Program
    {
        static int max = int.MinValue;
        static void fun(int[ ] arr, int low)

```

```

{
    if (low < arr.Length)
    {
        if (arr[low] > max)
            max = arr[low];
        fun(arr, ++low);
    }
    else
        Console.WriteLine(max);
}
static void Main(string[ ] args)
{
    int[ ] x = {-123,-565,-3555,-293,234,987,265};
    fun(x, 0);
}
}
}

```

الناتج :

987

في هذا المثال عرفنا متغير يسمى `max` ووضعنا به قيمة ابتدائية وهذه القيمة تأتي من خلال (`int.MinValue`) وتكون هذه القيمة هي أصغر قيمة يمكن أن يحملها متغير من النوع `int`، يُمكنك وضع قيمة ابتدائية صفر لكن بافتراض أن الأعداد المتواجدة في المصفوفة كلها سالبة ففي هذه الحالة لن يكون هناك رقم سالب أكبر من الصفر ويكون الناتج صفر وهذه النتيجة خاطئة، ثم أنشأنا دالة `void` وكان من الممكن أن تكون `int` ثم نُعيد قيمة الـ `max` من خلال كلمة `return`، وهذه الدالة تأخذ مصفوفة نوعها `int` وتأخذ أيضاً أصغر `index` في المصفوفة التي سنعمل عليها ودائماً ما يكون الصفر، ونكتب الشرط الذي نريد أن تتوقف الدالة عنده، متى تتوقف الدالة؟! عندما نصل إلى آخر `index` في المصفوفة أي أن المتغير الذي يسمى `low` عندما يصل إلى آخر `index` فإنها تتوقف بعد هذه العملية. فمثلاً إذا كان حجم المصفوفة 7 فتكون آخر قيمة

يصل إليها المتغير low هي 6 ، ثم بعد ذلك نتحقق من القيم الموجودة بالمصفوفة ما إذا كانت أكبر من قيمة الـ max أم لا، وأي قيمة أكبر من القيمة الحالية للـ max إذا ستكون قيمة الـ max هي هذه القيمة الجديدة ثم نستدعي الدالة داخل نفسها لكن نجعل المتغير low يزيد بمقدار 1 حتى ينتقل إلى الـ index الذي يليه وهكذا إلى أن يكون الشرط الخارجي خطأ وبذلك تتوقف الدالة عن استدعاء نفسها ويتم تنفيذ ما بداخل else وتطبع آخر قيمة وصل إليها المتغير max.

مثال :

أنشيء دالة تأخذ مصفوفة كمدخل وتخرج أكبر قيمة بهذه المصفوفة.

```
using System;
namespace ConsoleApplication1
{
class Program
{
public static int Max_Value_in_array ( int [ ] arr )
{
int max=arr[0] ;
for(int i = 0 ; i < arr.Length ; i++)
arr[i]=int.Parse(Console.ReadLine( ));
for( int i =0 ; i < arr.Length ; i++)
if ( arr [ i ] > max )
max = arr[ i ];
return max ;
}
static void Main( )
{
int [ ]x=new int[10] ;
Console.WriteLine ( Max_Value_in_array ( x ) ) ;
}
}
```

```
}
}
```

يمكن للدالة أن تأخذ مصفوفة ويتم التعامل معها بطريقة عادية جداً. يتم إدخال المصفوفة كـ (Parameter) مُدخِل للدالة، في هذه الحالة نحن لا نعلم حجم المصفوفة لذا توجد خاصية لإيجاد حجم المصفوفة **.Length**. فنستخدمها وحينما ننشئ المصفوفة في دالة الـ Main ونمررها للدالة يجب أن نحدد الحجم وحينها هذه الخاصية ستحسب حجم المصفوفة في الدالة بعدها تُمرر اسم المصفوفة التي أنشأتها في دالة الـ Main كمدخل للدالة، تُمرر اسم المصفوفة فقط.

تمارين :

أنشئ دالة تأخذ مصفوفتين وتنسخ أحدهما في الأخرى.

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Copy_array ( int [ ] arr ,int [ ] arr2)
        {
            for(int i = 0 ; i < arr.Length ; i++)
                arr2 [ i ] = arr [ i ] ;
        }
        static void Main( )
        {
            int [ ]x = new int[10]{ 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 } ;
            int [ ]y=new int[ 10 ] ;
            Copy_array ( x , y ) ;
        }
    }
}
```

18

(Object , Var , Dynamic)

يوجد ثلاثة أنواع من البيانات يمكنهم إستقبال أي بيانات من أي نوع، سواء كانت البيانات رقم صحيح أو كسر أو نص أو حرف أو أي نوع آخر، لكن يوجد فرق بين الثلاثة أنواع.

أول نوع هو :

object

كل أنواع البيانات هي عبارة عن (object) فيمكنه أن يأخذ بيانات من أي نوع.

وهو الـ class الرئيسي لكل أنواع البيانات ويكون في قاعدة هرم أنواع البيانات.

لكن (object) لا يفهم معنى البيانات التي بداخله، فلا يمكنه إجراء أي عملية على ما بداخله مع متغير آخر.

مثال :

```
object x = " aaa " ;
```

```
string y = x → خطأ
```

لا يمكن لأي نوع بيانات آخر من الأنواع المعروفة أن يحمل قيمة متغير من النوع object لأن نوعه يعتبر غير معلوم، هو فقط يحمل بيانات بداخله لكن لا يعلم نوعها. فلا يمكنه التعامل إلا مع object مثله. لكن ماذا لو أردنا تخزين القيمة التي تأتي من متغير نوعه object يجب في ذلك الوقت أن نحولها لنفس النوع الذي ستخزن فيه، فمثلا لو أردنا تنفيذ نفس العملية السابقة سنضطر إلى تحويل الـ x إلى نفس نوع الـ y.

```
object x = " aaa " ;
```

```
string y = ( string ) x ;
```

لكن يمكننا فعل العكس بدون تحويل.

```
string y = " aaa " ;
```

```
object x = y ;
```

لأننا أتفقنا أن الـ object يمكنه أن يحمل أي نوع آخر.

ويقبل أن نغير نوع البيانات التي بداخله بدون أن يحدث خطأ.

فمثلاً :

```
object x = " aa ";
```

```
x = 4;
```

```
Console.WriteLine( x ) ;
```

النتيجة :

4

يمكن أن يكون القيمة العائدة من دالة من النوع object إذا كنا لا نعرف النوع الراجع تحديداً.

ويمكن للمُدخل أيضاً أن يكون من النوع object. في حالة أننا لا نعرف القيمة التي سيدخلها للدالة هل هي نص أم حرف أم رقم.

```
using System;
namespace ConsoleApplication1
{
class Program
```

```

{
    public static object fun ( params object [ ] arr )
    {
        return arr[3] ;
    }
    static void Main( )
    {
        Console.WriteLine ( fun ( 3 , 4 , "aa" , 's' , true , 8 , 1 , 3.44
            , 3 , 4 , 5 ) );
    }
}
}

```

النتاج :

S

المصفوفة نوعها object لذا يمكنها أن تحتوي على أي نوع بداخلها، والدالة تعيد قيمة من نفس نوع المصفوفة وبما أن المصفوفة نوعها object فالقيمة العائدة منها إذاً ستكون من النوع object .

أكتب دالة لطباعة هذه المصفوفة التي أدخلنا فيها البيانات سابقاً.

```

using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void fun ( params object [ ] arr )
        {
            foreach( object i in arr )
                Console.WriteLine ( i ) ;
        }
        static void Main( )

```

```

    {
        fun ( 3 , 4 , "aa" , 's' , true , 8 , 1 , 3.44 , 3 , 4 , 5 ) ;
    }
}
}

```

is

هي كلمة محجوزة للتأكد من نوع البيانات .

مثال :

```

object x =10 ;
objext y = " aa " ;
if ( x is int )
{
    Console.WriteLine(" Yes ");
}

```

في هذا المثال سينفذ جملة الطباعة التي بداخل if لأن الشرط صحيح.

أما إذا قولنا :

```

if ( x is string )
{
    Console.WriteLine(" Yes ");
}

```

فلن ينفذ لأن المتغير x عدد صحيح وليس نصاً، إذاً فالكلمة المحجوزة is تتأكد من نوع البيانات هل هو متطابق أم لا.

var

هي مثل object لكن يوجد بعض الاختلافات بينهم.

فمثلاً عند تعريف متغير نوعه var لا يمكن تغيير القيمة التي بداخله لنوع آخر.

مثال :

```
var x = 3 ;
```

خطأ ; x = " aaa " ;

هو يقبل نوع واحد فقط فعندما مررنا له القيمة ٣ فهو يتعامل مع المتغير على أنه int ويجب أن نمرر له قيمة أثناء تعريف المتغير.

فلا يجوز أن نعرف المتغير من النوع var بدون تمرير قيمة له في مرحلة التعريف.

خطأ ; var x ;

يمكنه أن يأخذ أي قيمة. لكنه يتميز عن أن object في أنه عندما نُعرف متغير من نوعه فإنه يعرف نوع القيمة التي بداخله ويمكننا أن ندخلها في متغير آخر من نفس نوعها.

مثال :

```
var x = " aaa " ;
```

```
string y = x ;
```

ويمكننا إجراء أي عملية عليه طالما أننا نستخدمه مع متغيرات من نفس نوعه. لكن إذا أدخلناه في متغير من نوع آخر فيجب أن نحول نوعه أولاً لنفس النوع الجديد.

مثال :

```
var x = 3.4f ;
```

```
int y = ( int ) x;
```

ملاحظة : القيم الكسرية يكون نوعها تلقائياً `double`، إذا أردناها أن تكون `float` فإننا نكتب حرف **f** بجانب القيمة.

لا يستخدم لتعريف الدوال أو حتى المصفوفات، فهو يستخدم لتعريف المتغيرات فقط بداخل الدوال.

dynamic

هو عبارة عن نوع بيانات يستطيع أن يخزن أي شيء بداخله وتستطيع أن تغير قيمته لأي نوع تريده وأن تدخله في أي عملية تريدها. هذا النوع لا يتعرف على ما بداخله إلا وقت التشغيل.

```
dynamic c = " ttt " ;
string s = c + " aaa " ;
c = 6 ;
int n = c-3 ;
Console.WriteLine( s + " " + n );
```

الناتج :

```
tttaaa 3
```

حتى إذا وُجد خطأ فإنه لا يتعرف عليه إلا عند التشغيل لأنه يترجم ما بداخله في وقت التشغيل فقط.

مثال :

```
dynamic x = ' a ' ;
string y = x ;
```

لن يظهر الخطأ في الكود لكنه سيظهره عند تشغيل البرنامج.

يمكننا أن نستخدمه في تعريف دالة أو مصفوفة بدون مشكلة.

تمارين :

أكتب دالة لتسجيل عملية دخول إلى نظام أو موقع له بريد إلكتروني وكلمة سر

بافتراض أن البريد الإلكتروني المُسجل على قاعدة البيانات هو
Mahmoud@gmail.com

وكلمة السر هي mahmoud12345

```
using System;
namespace ConsoleApplication1
{
    class Program
    {
        public static void Login ( string _Email , string _Password )
        {
            if ( _Email == " Mahmoud@gmail.com " && _Password
                == " mahmoud12345 ")
                Console.WriteLine(" welcome ");
            else
                Console.WriteLine( " invalid data ");
        }
        static void Main( )
        {
            string E_Mail = Console.ReadLine ( );
            string password= Console.ReadLine ( );
            Login ( E_Mail , password );
        }
    }
}
```

19

(Classes)

ذكرت مسبقاً نبذة عن البرمجة كائنية التوجه وعن ماهية الـ class سنتعلم كيف نُنشئ الـ class.

عندما نُنشئ class فإننا نُنشئ نوع من البيانات التي نستطيع أخذ عدد غير محدود من النسخ من هذا النوع الواحد، وقد شرحت مثال لموقع الفيس بوك، فمثلاً صفحتك الشخصية تكون عبارة عن class به كل بياناتك الشخصية، وعند تسجيل الدخول يكون class به دوال لتنفيذ عملية الدخول، كل شخص يكون له بياناته الخاصة أي أن كل شخص قد أخذ نسخة من هذا الـ class وملاها ببياناته الخاصة.

إذا كنت تريد إنشاء برنامج أو موقع ويوجد عليه مستخدمون وكل مستخدم له بيانات بأنواع مختلفة مثل العمر والاسم وتاريخ الميلاد...، كيف ستربط كل هذه البيانات بمستخدم واحد؟!

هذه التقنية جعلت البرمجة عامةً سهلة ومرنة.

توجد قاعدة في البرمجة الكائنية تقول أن كل شيء هو عبارة عن كائن.

طريقة التعريف :

نكتب كلمة class ثم اسم الـ class، يُمكن أن نسميه بأي اسم نريده، لكن يُفضل أن يكون اسم يُعبر عما يقوم به هذا الـ class

```
class Name
```

```
{
}
```

ويُكتب بداخل namespace، طريقة الوصول لأي class تكون internal تلقائياً ويُمكنك تغييره إلى public، والبيانات التي تكون بداخل ال class تكون private تلقائياً ما لم نغيرها إلى نوع آخر.

إذاً ماذا نكتب بداخل ال class؟

نكتب بداخله كل الدوال التي نحتاجها لتنفيذ عملية معينة، والمتغيرات التي سنستخدمها، المتغيرات التي بداخل ال class غالباً ما تكون private ونستخدمها من خلال الدوال، ويكون نوع الدالة public. الطريقة المُتعارف عليها أن يُكتب أول حرف من اسم ال class بحرف كبير، مثل class Login

مثال :

```
using System;
```

```
namespace ConsoleApplication1
```

```
{
```

```
class First
```

```
{
```

```
}
```

```
class Program
```

```
{
```

```
static void Main( )
```

```
{
```

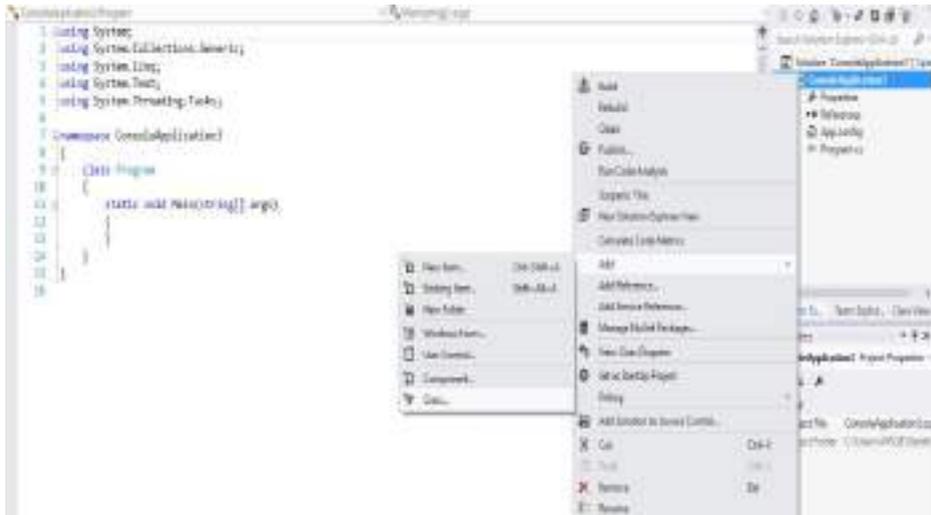
```
}
```

```
}
```

```
}
```

يمكننا إنشاؤه عن طريق كتابته مباشرةً كما فعلت، لكن يُفضل أن يكون كل class في صفحة مستقلة لكي يكون شكل الكود منظم وسهل في القراءة والتعديل.

نذهب إلى اسم المشروع على الفيجوال ستوديو ثم نضغط ضغطة يمين بالمؤشر فتظهر قائمة بها كلمة Add ونضغط على class ونسميه ونضغط enter، هكذا نكون قد أنشأنا class.



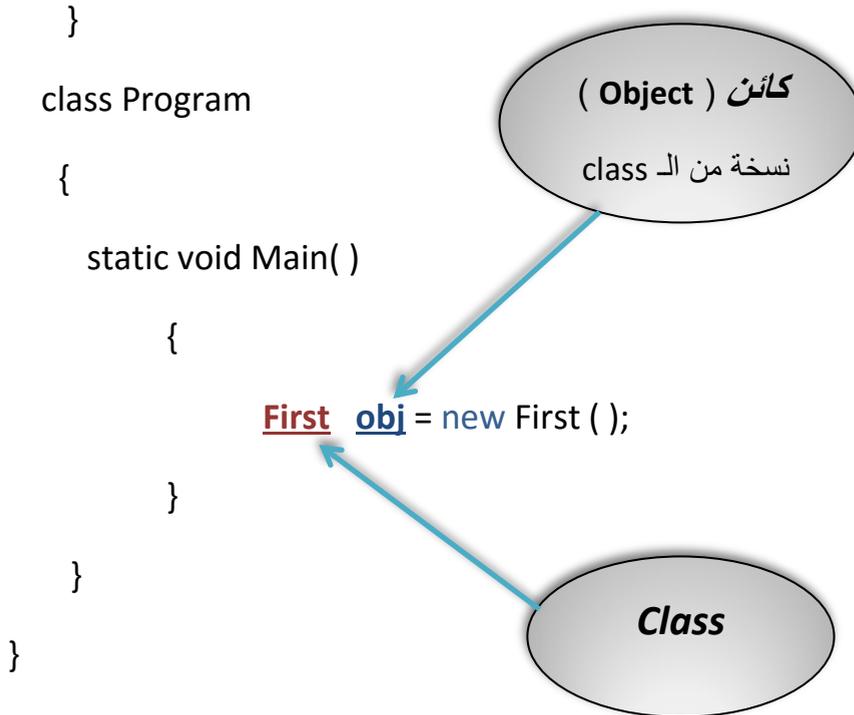
(١)



(٢)

لكي نأخذ نسخة من هذا ال class بكل البيانات التي بداخله " المسموح الوصول إليها فقط " نكتب اسم ال class ثم اسم المتغير، لكن في هذه الحالة يسمى كائن (object) وليس متغير.

```
using System;
namespace ConsoleApplication1
{
    class First
    {
        public int age ;
        string name ;
    }
}
```



تستخدم الكلمة المحجوزة **new** لخلق كائن جديد، واسم الـ class يتبعه قوسين يسمى constructor (" مُشيد ") سنعرف ماهو المُشيد لاحقاً، كل ما عليك معرفته مؤقتاً أن هذه طريقة تعريف الـ class وأخذ نسخة منه أو إنشاء كائن جديد منه.

عندما نُنشئ كائن (object) من class فإنه في ذلك الوقت يحتوي على كل خواص هذا الـ class المسموح له رؤيتها.

- نستطيع الوصول إلى كل خصائص الـ class من خلال "اسم الكائن (object) متبوعاً بنقطة " (.)

بعدها يمكننا أن نمرر له القيمة.

مثال :

أكتب class لموظف يحمل البيانات الآتية :
 (" الاسم ، العمر ، المرتب ، رقم الموبايل ").

```
using System;
namespace ConsoleApplication1
{
class Employee
{
    public string Name;
    public int Age;
    public int Salary;
    public string Mobile_number;
}
class Program
{
    static void Main( )
    {
        Employee emp = new Employee( );
        emp.Name = " ali " ;
        emp.Age = 30 ;
        emp.Salary = 3000 ;
        emp.Mobile_number = " +201111111111 " ;
    }
}
}
```

أولاً أنشأنا class بعد ذلك عرفنا الخصائص التي بداخله وإذا أردنا رؤيتها خارج هذا الـ class يجب أن يكون نوعها public كما عرفنا مسبقاً،

ثم أخذنا نسخة "كائن" (object) من هذا الـ class ومررنا قيم لهذه البيانات المجردة. كما اتفقنا البيانات التي تكون بداخل الـ class تكون مجردة أي أنها لا تحتوي على قيمة.

this

الكلمة المحجوزة **this** تستخدم بداخل الـ class وتستطيع رؤية كل ما بداخل الـ class وتعمل كأنها كائن من هذا الـ class فنستخدمها بداخل الدوال التي بداخل الـ class لكن غير مسموح إستخدامها بداخل دوال من النوع **static** ولا تستطيع رؤية المتغيرات أو الحقول التي تكون من النوع **static**.

سُعيد نفس المثال السابق لكن بإستخدام الدوال.

```
using System;
namespace ConsoleApplication1
{
class Employee
{
    string Name ;
    int Age ;
    int Salary ;
    string Mobile_number ;
    public void add_employee ( string name , int age , int salary ,
    string mobile )
    {
        this.Name = name ;
        this.Age = age ;
        this.Salary = salary ;
    }
}
```

```

        this.Mobile_number = mobile ;
    }
}
class Program
{
    static void Main( )
    {
        Employee emp = new Employee( );
        emp.add_employee ( "ali", 30 , 5000 ,
            "+201111111111");
        // لإضافة موظف آخر
        emp.add_employee ( "john" , 34 , 6000
            ,"+201222222222");
        //وهكذا
    }
}

```

أنشأنا دالة ويجب أن يكون نوعها `public` لكي تُرى خارج الـ `class` ولكي نستطيع التعامل معها، وبها المُدخلات التي سنمرر قيم من خلالها لخصائص الـ `class` أي للموظف كما في المثال.

بعدها أنشأنا كائن من هذا الـ `class` ثم أستدعينا الدالة التي تضيف البيانات ومررنا لها القيم بشكل مباشر. يمكنك أن تأخذ البيانات من المُستخدم بأن تعرف متغيرات في الدالة الرئيسية ثم تأخذ قيم لها من المستخدم ثم تمرر هذه المتغيرات للدالة.

```

string name = Console.ReadLine( );
int age = int.Parse(Console.ReadLine( ));
int salary = int.Parse(Console.ReadLine( ));
string mobile = Console.ReadLine( );
emp.add_employee ( name , age , salary , mobile );

```

وهكذا....

(المُشيد) Constructor

هو عبارة عن دالة ويكون بنفس اسم الـ class لكنه لا يرجع أي بيانات أي لا يكون له قيمة عائدة " Return value " مثل int أو String ولا حتى void. تعمل هذه الدالة تلقائياً عندما نأخذ نسخة من هذا الـ class يُكتب في هذه الدالة قيم ابتدائية نريد أن نمررها لمتغير ما ومن الممكن أن نمرر له مدخلات أثناء الإنشاء كالدالة تماماً، ويوجد أكثر من نوع للـ constructor.

أول نوع هو :

١- الـ **default** " التلقائي " :

وهذا النوع لا يأخذ أي مدخلات أثناء الإنشاء ويمكننا إعطاء قيم ابتدائية للحقول التي تكون في الـ class، وحتى لو لم نكتب الـ default constructor في الـ class فإنه يُنشأ تلقائياً، ويعطي للحقول التي من النوع int قيمة ابتدائية صفر وللحقول التي من النوع string قيم بتدائية بـ null، و null تُعبر عن اللا شيء.

طريقة تعريف المُشيد التلقائي :

```
public + Class_Name( )
{
}
}
```

مثال :

```
class Test
{
    public int id ;
    public string name;
    public Test( )
    {
        this.id = 1;
        this.name = "ali";
    }
}
```

```

}
}
class Program
{
    static void Main( )
    {
        Test T = new Test( ); // Default constructor.
        Console.WriteLine("My id is "+T.id + "and my name is " +
            T.name );
        // My id is 1 and my name is ali
    }
}

```

٢- مُشيدٌ له مُدخلات (Parameterized constructors)

وهو يأخذ مُدخلات مثل الدالة تماماً وعند أخذ نسخة من هذا الـ class فيجب تمرير مُدخلات له حتى يتم تنفيذه.

طريقة تعريف المُشيد الذي يأخذ مُدخلات :

```

public + Class_Name( المُدخلات )
{

}

```

مثال :

```

class Employee
{
    string Name ;
    int Age ;

```

```

int Salary ;
string Mobile_number ;
public Employee( ) // default constructor
{ }
public Employee( string name , int age )
{ // Parameterized constructors
    Name = name ;
    Age = age ;
}
public void add_employee ( string name , int age ,
                           int salary,string mobile )
{
    this.Name = name ;
    this.Age = age ;
    this.Salary = salary ;
    this.Mobile_number = mobile;
}
}

```

هذا المُشيد عندما تأخذ نسخة من الـ class فإنه يعمل تلقائياً ويطلب منك أثناء تعريف الكائن أن تضع قيم إبتدائية للبيانات التي أدخلتها في المُشيد أثناء إنشاؤه لكن إذا لم تدخل له مدخلات فلن يطلب منك أي شيء، وبالطبع هذه القيم الإبتدائية هي مجرد قيم مؤقتة يمكنك تغييرها، فمثلاً إذا استدعيت الدالة التي تُدخل البيانات فالبيانات الجديدة التي أدخلتها هي التي ستُخزن، البيانات التي بداخل المُشيد هي مؤقتة وتُخزن فقط في حالة أنك لم تدخل أي بيانات أخرى للمتغير.

```
Employee Emp = new Employee ( " Mahmoud ", 30 ) ;
```

هكذا يكون شكل المُشيد أثناء إستدعاؤه أو أثناء تعريف كائن جديد.

(Copy Constructor) -٣

وهذا النوع وظيفته هي أنه ينسخ قيم من (Object) كائن إلى (Object) كائن آخر من نفس الـ class، وأثناء تعريفه يأخذ object كائن أو نسخة من الـ class كمدخل.

مثال :

```
using System;
namespace Copy_Constructor
{
    class Program
    {
        static void Main(string[] args)
        {
            employee e = new employee(10, "John", 3000);
            employee em = new employee( e );
            em.print( );
            // Employee id : 10 , Employee name : John ,
            // Employee salary : 3000
        }
    }
    class employee
    {
        int id;
        string name;
        int salary;
        public employee(int _id, string _name, int _salary)
        {
            this.id = _id;
            this.name = _name;
            this.salary = _salary;
        }
    }
}
```

```

public employee(employee emp)
{
    this.id = emp.id;
    this.name = emp.name;
    this.salary = emp.salary;
}
public void print()
{
    Console.WriteLine(" employee id is : "+id+" \n Employee name :
    "+name+" \n Employee salary : "+salary);
}
}
}

```

(الهدام) Destructor

الهدام عكس المُشيد، هو أيضاً عبارة عن دالة تعمل تلقائياً بعد إنتهاء البرنامج، ووظيفته أنه يهدم أي كائن لم يُستخدَم أو كائن أُستخدِم لكن لم يعد له حاجة أو كائن لم يُستخدَم، وهو ليس كالمُشيد تحتاج أن تستدعيه. هو يعمل فور إنتهاء البرنامج دون إستدعاء ولا يحتاج إلى (Access modifier)، وكل class يكون له destructor واحد فقط.

طريقة التعريف :

```
~ Class_Name ( ) { }
```

هذه العلامة (~) تُسمى تيلدا .

مثال :

```
class Employee
```

```
{
```

```
string Name;
int Age;
int Salary;
string Mobile_number;
public Employee( ) // default constructor
{ }
public Employee( string name , int age )
{
    Name = name ;
    Age = age ;
}
public void add_employee ( string name , int age , int salary ,
string mobile )
{
    this.Name = name ;
    this.Age = age ;
    this.Salary = salary ;
    this.Mobile_number = mobile ;
}
~Employee ( )
{
    GC.Collect( ) ;
}
}
```

GC.Collect يشير إلى <--- Garbage Collector الذي تحدثنا عنه في بداية الكتاب في الجزء الخاص بإطار العمل.

20

الخصائص (Properties)

get , set

الموصلات (Accessors)

الحقول التي تكون بداخل الـ class تكون محمية ومخفية على المستخدم ولا يستطيع الوصول إليها، لكن ماذا لو كنا نريد تمرير قيم لهذه الحقول من خارج الـ class بدون أن نتعامل مع الحقول بشكل مباشر أو بدون دوال؟

في هذا الوقت سنستخدم set و get وهما عبارة عن كلمتين محجوزتين في اللغة حيث تُمكنك set من وضع قيم لهذه الحقول أي للكتابة فقط، و get تُمكنك من قراءة البيانات التي تكون بداخل الحقل أي تكون للقراءة فقط.

يوجد ثلاثة طرق للوصول إلى هذه الموصلات :

Public

Private

Protected

الموصلات تعامل معاملة الدوال.

فنحن نتحكم من يستطيع تمرير قيم للحقول ومن لا يستطيع، من يمكنه رؤيتها ومن لا يمكنه.

وُستخدم set في تحديد القيمة التي يُدخلها المستخدم، فمثلاً إذا كنت ستُدخل قيمة لمرتب لكن هذه القيمة غير منطقية مثل مرتب موظف ١ جنية في الشهر فأنت يمكنك أن تتحكم في مدى القيم الداخلة ويجب أن يُدخل المستخدم قيم في المدى المطلوب لكي تُخزن بداخل الحقل " المتغير في الـ class " فالقيم تأتي أولاً عن طريق set وإذا كان الشرط خطأً فلن تُخزن في الحقل، يُمكنك تحديد شرط قبل إستقبال القيم بداخل set أو لا تضع شروط، فالشرط ليس إضطرارياً لكن للتأكد من القيم فقط قبل تخزينها.

مثال :

```
class Employee
{
    private string Employee_Name ;
    private int Employee_Age ;
    private int Employee_Salary ;
    public string Name
    {
        set
        {
            Employee_Name = value;
        }
        get
        {
            return Employee_Name;
        }
    }
    public int Salary
    {
        set
        {
            if ( value <1000 )
            {
                Console.WriteLine ( " invalid data " );
            }
        }
    }
}
```

```

else
{
    Employee_Salary = value ;
}
}
get
{
    return Employee_Salary ;
}
}
}

```

معنى الكود أن :

أولاً أنشأنا class ثم عرفنا فيه الحقول (" الأسم ، العمر ، المرتب ") وطرق الوصول إليهم `private`، ثم أنشأنا خصائص لكي نتعامل مع هذه الحقول ونمرر لها قيم أو نقرأ منها هذه القيم.

الخصائص (" الأسم ، المرتب ") يتم إنشاء الخصائص تبعاً للحقول التي نريد للمستخدم التعامل معها بشكل غير مباشر فننشئ خاصية لكل حقل على حدة، وتسمى الخاصية بأي اسم نريده لكن يُفضل بأن تكون باسم قريب من اسم الحقل لكي نميزها.

ثم ننشئ الموصلات (`set , get`)، ولا يشترط كتابة `set` و `get` معاً في نفس الخاصية، بل يمكننا جعل هذا الحقل للقراءة فقط وعندها نستخدم `get` فقط، أو للكتابة فقط فنستخدم `set` أو كليهما.

نمرر القيمة للحقل من خلال كلمة `value` وهي كلمة محجوزة في اللغة تأخذ القيمة التي تأتي من المستخدم وتمررها للحقل المشار إليه ويجب أن تكون القيمة من نفس نوع الحقل او المتغير.

في خاصية الـ `Salary` أنشأنا شرطاً لكي نتأكد من القيمة التي تأتي من المستخدم هل هي في المدى المطلوب وفي حالة كانت في المدى المطلوب تُخزن بداخل الحقل أما إذا كانت خارج المدى فلا تُخزن.

وفي `get` نسترجع القيمة التي بداخل الحقل من خلال الكلمة المحجوزة `return`.