
# مقدمة في البرمجة
# Introduction To Programming

**IPG101**

## الفصل الخامس

## المكونات الأساسية للغة C#

## C# Fundamental Components

## الكلمات المفتاحية

نمط بيانات، متحول، ثابت، معامل، أسبقية معاملات، تحويل ضمني، تحويل صريح، إدخال، إخراج.

## ملخص الفصل

يدخل هذا الفصل في تفاصيل البرمجة في لغة C#، حيث يقدم تأسيساً يعرف بأهم المكونات الأساسية للغة من أنماط بيانات ومتحولات وثابت ومعاملات، ويشرح أسلوب التعامل معها ويعرف بمفهوم أسبقية المعاملات في التنفيذ ويوضح أساليب تحويل أنماط البيانات، كما يبين طرق إدخال وإخراج البيانات، ويختتم بتقديم بعض البرامج التسلسلية التي تتضمن تعابير برمجية بسيطة.

## أهداف الفصل

**بنهاية هذا الفصل سيكون الطالب قادراً على:**

- معرفة أنماط البيانات في لغة C#.
- استعمال الأنماط البسيطة.
- التصريح عن المتحولات والثوابت.
- كتابة تعابير برمجية باستخدام معاملات اللغة.
- إجراء عمليات الإدخال والإخراج من وإلى البرنامج.
- كتابة برامج بسيطة تسلسلية.

## محتويات الفصل

1. المتحولات والثوابت.
2. أنماط البيانات في لغة C#.
3. تحويل الأنماط.
4. معاملات اللغة وأفضلياتها.
5. الإدخال والإخراج.
6. التعليقات.
7. تمارين وأنشطة.

## 1- المتحولات والثوابت.

### المتحولات

**المتحول variable** هو عبارة عن مكان محجوز في الذاكرة مخصص لتخزين قيم من نوع معين.

يمكن إنشاء المتحولات في لغة C# باستخدام أمر التصريح عن المتحولات الذي يأخذ الصيغة العامة التالية:

```
<data type> <vaiable name> [= <value>];
```

**حيث أن:**

- **data type** نوع البيانات (القيمة) التي يمكن أن يأخذها (تخزن) هذا المتحول.
- **variable name** اسم المتحول.
- **[ = value]** مقطع إختياري يمثل القيمة الأولية لهذا المتحول.

**مثال:**

```
int num = 10;
```

يقوم الأمر السابق بحجز مكان في الذاكرة يدعى num ومقداره 4byte مخصص لتخزين قيم من النوع الصحيح ويقوم بتخزين القيمة 10 فيه.

**فيما يلي بعض الملاحظات المهمة:**

1- تحدد الكلمة data type نوع البيانات الخاص بهذا المتحول وبالتالي طيف القيم التي يمكن أن يأخذها والحجم في الذاكرة الذي يتم حجزه لتخزين هذه القيمة (نفصل في هذا الأمر في الفقرة القادمة).

2- يحدد الجزء variable name اسم المتحول، وهو الاسم الذي نطلقة عليه ونقوم بمناداته بواسطته، يمكن أن نقوم بتسمية المتحول الاسم الذي نرغب به، مع مراعاة القواعد التالية:

a. إسم المتحول يجب أن يكون فريداً unique (أي لا يمكن تكراره ضمن نفس مجال الرؤية).
b. يمكن أن يحتوي إسم المتحول أحرفاً، أرقاماً، ومحرف الشرطة السفلية ( underscore ) فقط.
c. يجب أن تبدأ أسماء المتحولات بمحرف.
d. لا يمكن لاسم المتحول أن يكون من الكلمات المحجوزة في اللغة.

**ملاحظات حول اسم المتحول:**

- لغة C# هي لغة حساسة لحالة الأحرف case-sensitive وبالتالي الاسمان num و Num هما اسمان مختلفان.

- يستحسن تسمية المتحولات بأسماء دالة على وظيفتها (وخاصة عند تضخم البرنامج) لسهولة التعامل معها وفهم الشيفرة البرمجية، فمثلاً إذا أردنا تخزين قيمة السعر، فمن المستحسن تسمية المتحول بالاسم price (لأن استخدام إسم مثل x، y، ... إلخ) يجعل فهم البرنامج وتفسيره أمراً صعباً.
- يمكن استخدام الكلمات المحجوزة كأسماء لمتحولات إذا سبقناها بالمحرف @ (إلا أن هذا الأمر غير منصوح به).

3- لا يمكن البدء باستخدام المتحول في لغة C# قبل إعطائه قيمة، يقوم الجزء الاختياري [=value] بإعطاء قيمة للمتحول لحظة إنشائه (ويمكن حذف هذا الجزء وإعطاء المتحول قيمة ما لاحقاً).

أمثلة عن تعريف متحولات:

- تعريف متحولات بدون قيم أولية

```
int MaxN;
double Value2 ;
char aChar ;
bool Test ;
long N_F ;
```

- تعريف متحولات مع إسناد قيم أولية

```
int MaxN = 50 ;
double Value2 = 2.3;
char aChar = 'K' ;
bool Test = false ;
long N_F = 1600000000000;
```

**مفاهيم إضافية حول المتحولات:**

**أولاً –** من الممكن التصريح عن المتحول بدون قيمة أولية ومن ثم إعطائه قيمة متى رغبنا كما يلي:

```
int MaxN;
MaxN = 50;
```

**ثانياً –** لا يمكن البدء باستخدام المتحول أو الوصول إليه ما لم يتم إعطاؤه قيمة، أمثلة:

```
int MaxN;
int MinN = MaxN;                    // Error MaxN is not assigned.
int MinN = MaxN - 100;              // Error MaxN is not assigned.
Console.WriteLine(MaxN);            // Error MaxN is not assigned.
```

**ثالثاً** – يمكن التصريح عن أكثر من متحول من نفس النوع في سطر أوامر واحد بحيث تكون مفصولة فيما بينها بفواصل (سواء مع إعطاء قيم إبتدائية أو بدونها)، أمثلة:

```
int MaxN, MinN, FirstN;
int MaxN=100, MinN, FirstN=1000;
```

**رابعاً** – لغة C# هي لغة صارمة بما يتعلق بأنواع البيانات، وبالتالي لا يمكن إعطاء متحول قيمة لا تتناسب مع نوعه:

```
int MaxN  = "Hello";    // Error, cannot implicitly convert type 'string' to 'int'
```

## الثوابت

رأينا فيما سبق، أن المتحول يحجز مكاناً في الذاكرة لتخزين قيمة، وهذا المكان يمكن أن يتم تغيير قيمته في الوقت الذي نرغب به (بناء على معطيات المسألة)، ولكن ماذا لو احتجنا لحجز مكان في الذاكرة وتخزين قيمة ثابتة فيه لا يمكن تغييرها طوال البرنامج، في هذه الحالة يجب أن نشير إلى هذا المكان على أنه ذو قيمة ثابتة.

**الثابت Constant** هو عبارة عن مكان محجوز في الذاكرة يأخذ قيمة ثابتة لا يمكن تغييرها أبداً.

يمكن إنشاء الثوابت في لغة C# بإضافة الكلمة المفتاحية const إلى أمر التصريح عن متحول مع إعطاء قيمة لهذا المعرف لحظة إنشائه، وفق الصيغة العامة التالية:

```
const <data type> <vaiable name> = <value>;
```

**مثال:**

```
const int num = 10;
```

يقوم الأمر السابق بحجز مكان في الذاكرة يدعى num ومقداره 4byte مخصص لتخزين قيم من النوع الصحيح ويقوم بتخزين القيمة 10 فيه ولا يسمح بتغييرها أبداً في أي لحظة ضمن البرنامج.

يعتبر إعطاء القيمة الإبتدائية أمراً إلزامياً وضرورياً (بخلاف المتحولات) ولا يمكن تغييرها بعد ذلك، وبالتالي فإن الأوامر التالية تتضمن أخطاء:

```
const int MaxN;              // Error const field requires value to be provided.
const int MinN = 100;
MinN = 50;                   // Error The left-hand side of assignment must be a variable
```

## 2- أنماط البيانات في لغة C#

رأينا سابقاً كيف أننا قمنا لدى التصريح عن المعرفات (متحولات، ثوابت) بتحديد نمط (نوع) البيانات لهذا المعرف وعملنا أن لغة C# هي لغة صارمة لناحية التعامل مع أنماط البيانات.
نتعرف الآن على أنماط البيانات التي يمكن للغة C# أن تتعامل معها.

يمكن تصنيف أنماط البيانات في لغة C# **بحسب تعريفها** إلى:

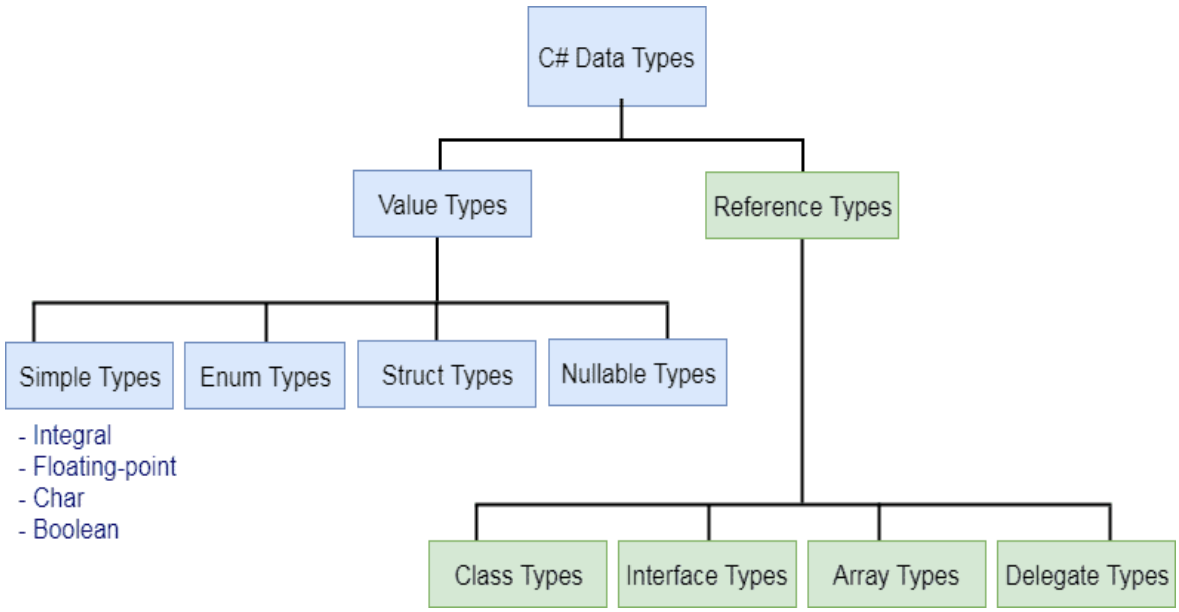
- **أنماط مسبقة التعريف predefined types** وهي معرفة مسبقاً في اللغة ويمكن استخدامها مباشرة من قبل المستخدم، من أمثلتها int,float,long,…… etc.
- **أنماط معرفة من قبل المستخدم user-defined types** وهي أنماط يعرفها المستخدم ومن ثم يصبح قادراً على استخدامها، من أمثلتها الأصناف classes، السجلات structs …. إلخ.

أما من حيث **التركيب**، فتصنف إلى:

- **أنماط بسيطة simple types**: وتأخذ قيمة واحدة، من أمثلتها int,float,long,…… etc.
- **أنماط مركبة compound types**: وتأخذ عدداً كبيراً من القيم من نفس النوع (أو من أنواع مختلفة)، من أمثلتها الأصناف classes، السجلات structs ، المصفوفات arrays، السلاسل المحرفية strings .. إلخ.

وأخيراً من حيث **القيم**، فتصنف إلى:

- **أنماط القيمة value types**: وتأخذ قيمة مباشرة.
- **أنماط المرجع reference types**: وتأخذ قيمة تشير إلى مكان في الذاكرة.




**الشكل 1- أنماط البيانات بحسب القيمة.**

نهتم حالياً بأنواع البيانات البسيطة، مسبقة التعريف، ذات القيمة، يبين الجدول التالي ملخصاً بهذه الأنماط:

| التصنيف | النمط | عدد الـ Byte التي يحجزها في الذاكرة | توصيف |
|---|---|---|---|
| الأعداد الصحيحة | **byte** | 1 | قيمة صحيحة بدون إشارة تتراوح بين 0 و 255 |
| | **sbyte** | 1 | قيمة صحيحة ذات إشارة تتراوح بين -128 و +127 |
| | **short** | 2 | قيمة صحيحة ذات إشارة تتراوح بين $-2^{15}$ و $2^{-15}-1$ |
| | **ushort** | 2 | قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{16}-1$ |
| | **int** | 4 | قيمة صحيحة ذات إشارة تتراوح بين $-2^{31}$ و $2^{31}-1$ |
| | **uint** | 4 | قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{32}-1$ |
| | **long** | 8 | قيمة صحيحة ذات إشارة تتراوح بين $-2^{63}$ و $2^{63}-1$ |
| | **ulong** | 8 | قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{64}-1$ |
| الأعداد العشرية | **float** | 4 | فاصلة عائمة بدقة بسيطة ~ 7 أرقام عشرية |
| | **double** | 8 | فاصلة عائمة بدقة مُضاعفة ~ 15 رقم عشري |
| | **decimal** | 16 | دقة تصل إلى 28 رقم عشري على الأكثر |
| المحارف والسلاسل | **string** | حسب عدد الأحرف | سلسلة محارف |
| | **char** | 2 | حرف يتبع الترميز Unicode (قيمته بين 0 و 65536) |
| النمط المنطقي | **bool** | 1 | يأخذ إحدى القيمتين TRUE أو FALSE |

**ملاحظات:**

- u هي اختصار لـ unsigned بلا إشارة (أي عدد موجب)
- لا يمكننا تصنيف النمط string (سلسلة محارف) كنمط بسيط، فهو مركَّب من مجموعة من الحروف، ولكنه وُضع في الجدول لأن الجدول يضم جميع الأنماط المتاحة في لغة C# . وجميع أسماء هذه الأنماط هي كلمات محجوزة keyword في لغة C#.

## 3- تحويل الأنماط

رأينا سابقاً أن لغة C# هي لغة صارمة لناحية التعامل مع أنماط البيانات، وبالتالي فإن الأوامر التالية تسبب الأخطاء المشار إليها بجانب كل أمر:

```
int x  = “Hello”;    // Error, cannot implicitly convert type ‘string’ to ‘int’
string name = 10.5; // Error, cannot implicitly convert type ‘double’ to ‘string’
```

تتيح لغة C# إمكانية القيام بتحويل أنماط القيم من نوع لآخر سواء بشكل صريح أو بشكل ضمني.

### التحويل الضمني Implicit Conversion

يمكن بشكل عام إسناد متغير من نمط إلى متغير من نمط آخر بشكل مباشر في حال كان مجال قيم المتغير الأول ضمن مجال قيم المتغير الثاني. مثلاً:

```
int i = 100;
long j;
j = i;
```

يبين الجدول التالي من أجل كل نمط الأنماط التي يُمكن الإسناد إليها بشكل ضمني:

| To | From |
|---|---|
| short, int, long, float, double, decimal | sbyte |
| short, ushort, int, uint, long, ulong, float, double, decimal | byte |
| int, long, float, double, decimal | short |
| int, uint, long, ulong, float, double, decimal | ushort |
| long, float, double, decimal | int |
| long, ulong, float, double, decimal | unint |
| float, doule, decimal | long, ulong |
| double | Float |

### التحويل الصريح Explicit Conversion

يمكن أيضاً إجراء عملية "قصر" بحيث نقوم بإسناد متغير من نمط ذو مجال قيم أكبر إلى نمط ذو مجال قيم أقل. بالطبع، يمكن ألا تعطي هذه العملية نتائج صحيحة في حال كانت القيم المسندة خارج مجال المتغير المسند إليه.

**أمثلة**

```
//Explicit Conversion
long v = 30000;
int i= (int) v ;              // A valid Cast. The max is 2147483647,  i will be 30000
long g=3000000000;    //An invalid Cast. The max is 2147483647, j will be -129947296
int j= (int) v;
int k= checked((int) g);              // throw an overflow exception if needed
double p=25.7;
int m= (int) p;                       // m will be 25
char c = 'A';
int i = (int)c;                        //i will be 65
int i = 10;
string s = i.ToString();              //s will be "10"
string ss = "300";
int j = int.Parse(ss);                 //j will be 300
string sss = "34.678";
double x = double.Parse(sss);      //x will be 34.678
int k = int.Parse(sss);               //Throw an exception
```

**بعض الملاحظات:**

- للتحويل الصريح من نمط إلى آخر يكفي أن نقوم بذكر النمط المحول إليه (محصوراً بين قوسين) قبل القيمة المراد تحويلها.
- يمكن التحويل من النمط string إلى الأنماط الرقمية (صحيحة وعشرية) باستخدام التابع parse العائد للصنف الخاص بالنمط المحول إليه ( Byte, SByte,Int16,UInt16,Int32,UInt32,Int64,UInt64, Single,Double )
- يمكن التحويل من النمط الرقمي (صحيح أو عشري) إلى سلسلة محرفية باستخدام التابع ToString.
- يمكن استخدام الطريقة checked لإطلاق استثناء في حال كانت عملية القصر لا تعطي نتائج صحيحة.

يمكن استخدام الصف Convert المسبق التعريف في اللغة لإجراء التحويلات الصريحة يلي:

```
long v = 30000;
int i = Convert.ToInt32(v);                    // same as int i= (int) v ;
```

بالمثل لتوابع الصنف Convert الأخرى ()ToByte، ()ToBoolean، ()ToInt64 .... إلخ.

## 4- معاملات اللغة وأفضلياتها

**المعامل operator** هو عبارة عن رمز ينفذ فعلاً معين.

هناك نوعان من المعاملات في لغة C#، **المعامل الأحادي unary operator** ويؤثر على وسيط واحد، و**المعامل الثنائي binary operator** ويؤثر على وسيطين إثنين (أحدهما عن يمينه والآخر عن يساره).

تمتلك لغة C# الأصناف التالية من المعاملات:

### المعاملات الحسابية Arithmetic Operators

تستخدم هذه المعاملات لإجراء العمليات الحسابية على وسطاء من جميع الأنواع الرقمية مثل sbyte, byte, short, ushort, int, uint, long, ulong, float, double, decimal

| المعامل | نوع المعامل | التوصيف | الأفضلية | مثال |
|---|---|---|---|---|
| + | أحادي | إشارة موجبة | 1 | +a; +b; +1; x=+z; |
| – | أحادي | إشارة سالبة | 1 | -a; -b; -(5+x); y=-(6*u); |
| ++ | أحادي | زيادة بقيمة واحد | 1 | a++; ++y; v=++c; |
| –– | أحادي | نقصان بقيمة واحد | 1 | x--; --z; x=--p; |
| * | ثنائي | عملية الضرب | 2 | (a*b); x*y; 6*z; x=a*b; |
| / | ثنائي | عملية القسمة | 2 | (a/b); x/5; y=u/y; |
| % | ثنائي | عملية باقي القسمة | 2 | x%5; z%a; z=u%t; |
| + | ثنائي | عملية الجمع | 3 | a+b; (x+y)+z; t=x+y; |
| – | ثنائي | عملية الطرح | 3 | -a-b; x-y-z; t=x-y-z; |

### معاملات الإسناد Assignment Operators

تقوم هذه المعاملات بإسناد القيمة التي إلى يسارها إلى الوسيط الذي إلى يمينها.

| المعامل | نوع المعامل | التوصيف | الأفضلية | مثال |
|---|---|---|---|---|
| = | أحادي | إسناد قيمة | أخيرة | x= 10; z=y; y=x+100; |
| Op = | أحادي | إسناد حسابي مكافيء للعملية: x = x op y | أخيرة | x +=5; z*=y; n%=2; t-=x; m/=4; |

## معاملات المقارنة Comaprison Operators

تقوم هذه المعاملات بمقارنة قيمتين وإعادة true أو false بحسب ناتج المقارنة.

| المعامل | نوع المعامل | التوصيف | الأفضلية | مثال |
|---|---|---|---|---|
| > | ثنائي | أكبر تماماً | 5 | a>b; a>(x+y); |
| >= | ثنائي | أكبر أو يساوي | 5 | a>=b; a>=(x+y); |
| < | ثنائي | أصغر تماماً | 5 | a<b; a<(x+y); |
| <= | ثنائي | أصغر أو يساوي | 5 | a<=b; a<=(x+y); |
| == | ثنائي | يساوي | 6 | a==b; (x+y) == (z+r); |
| != | ثنائي | لايساوي | 6 | a!=b; (x+y) != (z+r); |

## المعاملات المنطقية أو الثنائية Boolean Logical Operators

تقوم هذه المعاملات بإجراء العمليات على القيم الثنائية.

| المعامل | نوع المعامل | التوصيف | الأفضلية | مثال |
|---|---|---|---|---|
| ! | أحادي | نفي | 1 | ! ((a+b) < 6); |
| & | ثنائي | و | 7 | ((a+b)<6) & ((x+y)>7); |
| \| | ثنائي | أو | 9 | ((a+b)<6) \| ((x+y)>7); |
| && | ثنائي | "و" مُحسَّنة | 10 | ((a+b)<6) && ((x+y)>7); |
| \|\| | ثنائي | "أو" مُحسَّنة | 11 | ((a+b)<6) \|\| ((x+y)>7); |

## أفضلية المعاملات Operators Precedence

- يمكن للأقواس أن تحل مشكلة الأفضليات؛

- على سبيل المثال يكون للعبارة ((x+y) > z) && (z<8) التفسير التالي:
  - يجري أولاً حساب x+y ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة ((x+y) > z)؛
  - يجري بعدها مقارنة قيمة z بـ 8 لتحديد خطأ أو صحة العبارة (z<8)؛
  - يجري بعد ذلك التحقق من صحة أو خطأ ((x+y) > z) && (z<8) تبعاً لجدول الحقيقة الخاص بالعملية &&؛

- في حال عدم وجود أقواس يتم تنفيذ العمليات تبعاً للأفضليات (العمليات ذات الأفضلية 1 لها أسبقية على العمليات ذات الأفضلية 2 وهكذا دواليك)؛

- أما في حال تسلسل عمليتين لهما نفس الأفضلية، فتكون الأسبقية للعملية الموجودة على اليسار؛

- على سبيل المثال يكون للعبارة (x+y > z && z<8) التفسير التالي:
  - بما أن عملية "الجمع" تمتلك أسبقية (ذات الأفضلية 3) بالنسبة لعملية المقارنة "أكبر تماماً" (ذات الأفضلية 5) يجري أولاً حساب x+y ومقارنة النتيجـــــــــة بقيمة z لتحديد خطأ أو صحة العبارة (x+y) > z؛
  - بما أن عملية المقارنة "أصغر تماماً" تمتلك أسبقية (ذات الأفضلية 5) بالنسبة لعملية الـ "و" المنطقية (ذات الأفضلية 10) يجري أولاً حساب z<8 ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة z<8؛
  - يجري بعد ذلك التحقق من صحة أو خطأ (x+y > z && z<8) تبعاً لجدول الحقيقة الخاص بالعملية &&؛

أمثلة:

| X | Y | Z |
|---|---|---|
| 5 | 3 | 2 |

| | T |
|---|---|
| `T = X + Y * Y;` | 14 |
| `T = X + Y * Y / Z;` | 9 |
| `T = (X + Y) * Y / Z;` | 12 |
| `T = (Y * Y) * (Y / Y + X);` | 54 |

## 5- الإدخال والإخراج

تستخدم لوحة المفاتيح (كدخل قياسي) في عملية إدخال (قراءة) القيم إلى البرنامج في لغة C#، وتستخدم الشاشة (كخرج قياسي) في عملية إخراج (كتابة) القيم والرسائل.

### إدخال (قراءة) القيم

لقراءة قيمة متحول ذو نمط بسيط ، يمكن أن نستخدم تعليمة *Read* أو *ReadLine* التابعة للصف Console وإسنادها للمتحول المطلوب، إلا أن القيمة التي ترجعها *Read* أو *ReadLine* هي من نمط سلسلة المحارف. فإذا أدخلنا 123 تمت قراءتها من قبل التعليمة على أنها سلسلة المحارف "123".

يؤدي استخدام التعليمة *ReadLine* دون أخذ الملاحظة الآنفة الذكر بعين الاعتبار إلى حدوث خطأ (عدم توافق الأنماط الناتج عن قراءة قيمة ذات نمط محرفي ومحاولة اسنادها لمتحول يعبر عن عدد صحيح) في حال نفذنا المقطع التالي:

```
int Num;
Num = Console.ReadLine();     // Error, Cannot implicitly convert from 'string' to 'int'
```

لذا يتوجب في حال أردنا أن نقرأ متحول من نمط عدد صحيح، أو حقيقي أن نستخدم إجرائيات تحويل خاصة كإجرائية Parse المرتبطة بكل نمط من الأنماط البسيطة والتي تقوم بتحويل سلسلة محارف مثل "123" إلى قيمة هي 123، كما هو الحال في المقطع التالي:

```
string s;
int Num;
s = Console.ReadLine();
Num = Int32.Parse(s);
```

للتنويه، كان من الممكن من باب الاختصار إنجاز كامل العملية السابقة في أمر واحد كما يلي:

```
int Num  = Int32.Parse(Console.ReadLine());
```

**ملاحظة هامة**

يمكن في مترجمات C# حالياً الاستغناء عن استخدام الصف المقابل (الصف المغلِّف) للنمط البسيط، فالمترجم يغلف النمط البسيط بدلاً عنا!

فتعليمات القراءة التالية صالحة (وأسهل للتذكر)

```
string s = Console.ReadLine();
int Num= int.Pars(s);
```

أو

```
string s = Console.ReadLine();
float f =float.Parse(s);
```

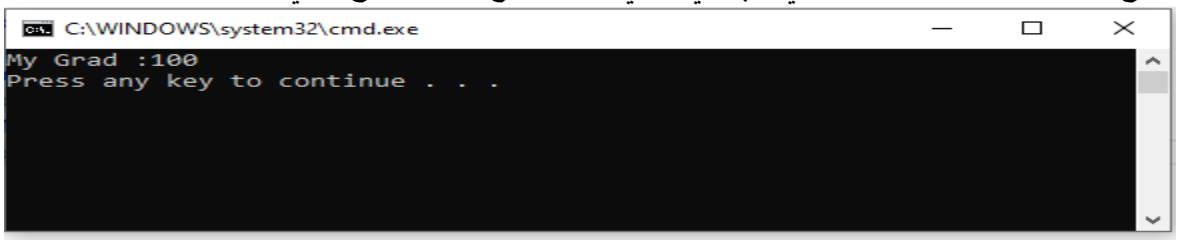
## إخراج (كتابة) القيم

لإخراج قيمة متحول ذو نمط بسيط، أو قيمة مباشرة (رقمية أو نصية) يمكن أن نستخدم تعليمة *Write* أو *WriteLine* التابعة للصف Console.

لإخراج قيمة متحول، يمكن أن نقوم بتمرير هذا المتحول إلى الأمر WriteLine() أو Write() كمايلي:

```
string Message = “My Grad :”;
Console.Write(Message);
int Num = 100;
Console.WriteLine(Num);
```

يقوم الأمر Write بإخراج القيمة الممررة إلى الشاشة مع البقاء ضمن نفس السطر، أما الأمر WriteLine فيؤدي إخراج القيمة والانتقال إلى السطر التالي، وبالتالي يعطي تنفيذ المقطع السابق الخرج التالي:




تسمح لغة C# باستخدام أي تعبير كقيمة ممررة إلى التابع WriteLIne وحينها سيتم تقييم التعبير ومن ثم إخراج ناتج التقييم إلى شاشة الخرج، كمثال ليكن المقطع التالي:

```
string Message = "My Grad :";
int Num = 100;
Console.WriteLine(Message+Num);
//Output: My Grad : 100
Console.WriteLine(Num * 5);
//Output: 500
Console.WriteLine(Num > 10);
//Output: True
```

كما يتولى المترجم تفسير محارف الاستبدال ( وهي محارف ذات مدلول خاص يمكن أن تضمن ضمن السلسة المحرفية مسبوقة بالمحرف \ ومن أمثلتها \n ، \t ، \' ، \" ، \a .... إلخ. ، كمثال ليكن المقطع التالي:

```
Console.WriteLine("Column 1\tColumn 2\tColumn 3");
//Output: Column 1        Column 2        Column 3

Console.WriteLine("Row 1\nRow 2\nRow 3");
/* Output:
   Row 1
   Row 2
   Row 3
*/
```

وفي حال الرغبة في إخراج السلسلة المحرفية كما هي بمحارفها الخاصة أو على أسطر متعددة ... إلخ، تتيح لغة C# استخدام المحرف @ قبل السلسلة المحرفية للدلالة على ذلك.

```
Console.WriteLine( @"C:\MyProject\Lesson1\TestWriteLine\");
 //Output: C:\MyProject\Lesson1\TestWriteLine\

Console.WriteLine( @"I Am a Student In SVU !
 My Current Course is IPG101
 My Current Lesson is C# ,...");
 /* Output:
  I Am a Student In SVU !
 My Current Course is IPG101
 My Current Lesson is C# ,...
   */
```

أخيراً من الممكن تضمين قيم المتحولات داخل السلاسل المحرفية من خلال وضعها ضمن {} وتمريرها كبارامترات للتابع WriteLine كمايلي:

```
string firstName = "Ahmad";
string lastName = "Ali";
string id = "29288827";
Console.WriteLine("My Name is {0} {1} My Id = {2}", firstName, lastName, id);
//output My Name is Ahmad Ali My Id = 29288827
```

## 6- التعليقات

**التعليقات comments** هي عبارة عن نصوص وعبارات تكتب داخل النص البرمجي ولايقوم المترجم بقرائتها أو ترجمتها.

تكتب التعليقات ضمن الشيفرة البرمجية لأغراض لها علاقة بالتوثيق وكتابة العبارات التوضيحية لقارئ البرنامج، حيث يمكن أن يكتب أي نص يرغب به المبرمج.

هناك نوعان من التعليقات يمكن إدارجهما ضمن النص البرمجي في لغة C# وهما:

**التعليق السطري :**

ويكتب التعليق بعد المحرفين // ويمتد حتى نهاية السطر كما يلي:

```
// Single Line Comment
```

**التعليق متعدد الأسطر:**

ويكتب بين الرمزين /* و */ ويمكن أن يمتد لعدة أسطر كمايلي:

```
/* First Line in The Comment
Secod Line in The Comment
…. Etc
*/
```

## 7- تمارين وأنشطة

**تمرين 1** – نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;
namespace HelloWorld3s
{
    class Welcome3
    {
        static void Main( string[] args )
        {
          Console.WriteLine( "Welcome\nto\nC#\nProgramming!" );
        }
    }
}
```

**تمرين 2**– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;
namespace AdditionProgram
{
    class Addition
    {
        static void Main( string[] args )
    {
         string firstNumber, secondNumber;
          int number1, number2,  sum;
          // prompt for and read first number from user as string
          Console.Write( "Please enter the first integer: " );
          firstNumber = Console.ReadLine();
          // read second number from user as string
          Console.Write( "\nPlease enter the second integer: " );
          secondNumber = Console.ReadLine();
          // convert numbers from type string to type int
          number1 = Int32.Parse( firstNumber );
          number2 = Int32.Parse( secondNumber );
          // add numbers
          sum = number1 + number2;
          // display results
          Console.WriteLine( "\nThe sum "+sum );
        } // end method Main
    } // end class Addition
} // end namespace AdditionProgram
```

**تمرين 3**– ماهي الأخطاء التي ستظهر عند ترجمة التعليمات التالية:

```
int val1, val2;
uint pos_val;
val1 = 1.5;
val2 = 9876543210;
pos_val = -123;
```

**تمرين 4**– ماذا ستكون قيمة m بعد تنفيذ كل من التعليمات التالية

```
int i = 10;
int j = 5;
int m;
m = i / j * j + i % j;
```

```
int i = 12;
int j = 5;
int m;
m = i / j * j;
```

**تمرين 5**– بفرض أن لدينا خزان ماء أسطواني الشكل.

فإذا علمت أن حجم الخزان يحسب كما يلي:

$V=\pi \times R^2 \times H$

حيث $\pi$ هي ثابت حقيقي مقداره 3.1415 و R نصف قطر القاعدة و H ارتفاع الخزان.

يطلب كتابة برنامج بلغة C# يقوم بالطلب إلى المستخدم إدخال قيمتي نصف قطر القاعدة والارتفاع ثم يقوم بحساب مقدار حجم الخزان وطباعته على الشاشة.

**تمرين 6**– أكتب برنامجاً لتحويل قيمة ما مقاسة بالقدم feet إلى القيمة المقابلة باليادرة yard، بالإنش inches، بالسنتيمتر centimeters، بالمتر meters.

حيث أن:

1 feet = 12 inche.
1 yard = 3 feet.
1 inche = 2.54 centimeters.
1 meter = 100 centimeters.