الجامعـة الافتراضيّـة السوريّـة
SYRIAN VIRTUAL UNIVERSITY

البرمجة الإجرائية

الدكتور باسل الخطيب

ISSN: 2617-989X

Books

البرمجة الإجرائية
باسل الخطيب

من منشورات الجامعة الافتراضية السورية

الجمهورية العربية السورية 2018

Procedural Programming

Bassel Al Khatib

Publications of the Syrian Virtual University (SVU)

Syrian Arab Republic, 2018

ISSN: 2617-989X

الفهرس

ISSN: 2617-989X

ISSN: 2617-989X

ISSN: 2617-989X

مقدمة في الصفوف والطرق باستخدام Visual Studio 2013

ISSN: 2617-989X

الكلمات المفتاحية:

التصريح عن صف، التصريح عن طريقة، إنشاء غرض من صف، استدعاء طريقة، الطرق الساكنة، الطريقة Main.

ملخص:

نُبين في هذا الفصل أساسيات البرمجة الإجرائية في محيط العمل Visual Studio 2013. حيث نعرض كيفية التصريح عن الصفوف والأغراض والطرق.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- التصريح عن الصفوف
- إنشاء الأغراض من الصفوف
- إنشاء واستدعاء الطرق
- الطرق الساكنة
- الطريقة الساكنة Main

المخطط:

يتضمن فصل "الصفوف والطرق باستخدام Visual Studio 2013" 3 وحدات عناوينها بالترتيب المحدد:

1. الصفوف والأغراض والطرق.
2. الطرق الساكنة.
3. الطريقة Main.

1. الصفوف والأغراض والطرق

الأهداف التعليمية:

إنشاء الصفوف والأغراض في Visual Studio 2013

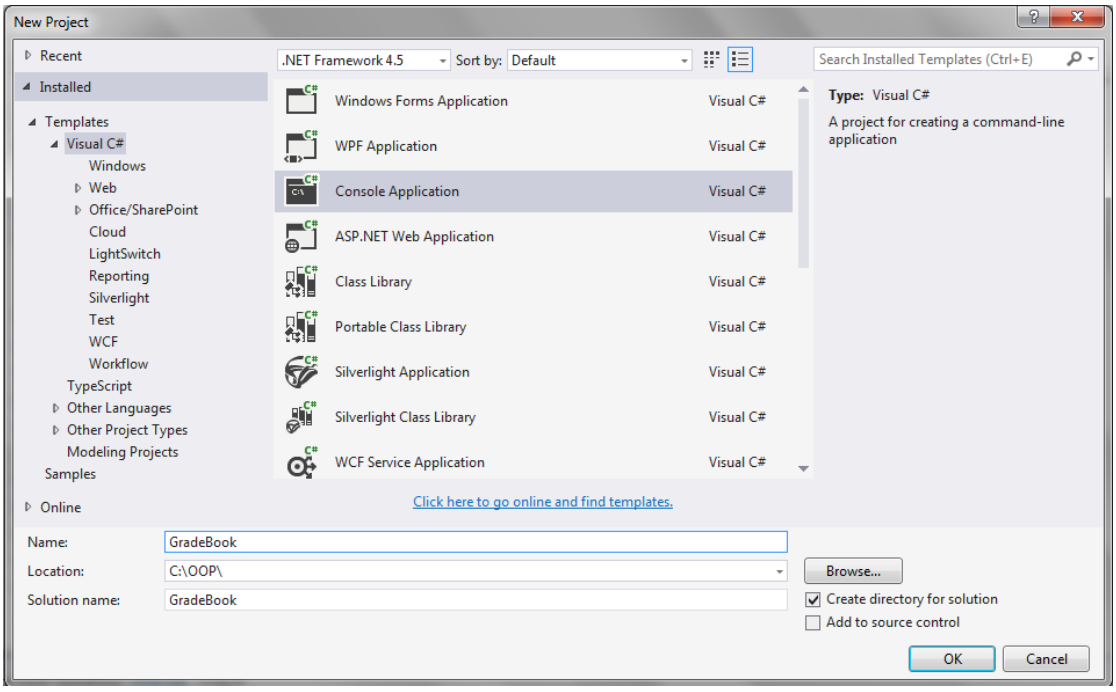
إنشاء صف يحوي طريقة وإنشاء غرض من الصف:

يسمح محيط العمل Visual Studio 2013 بإنشاء التطبيقات بشكل بسيط وسريع:

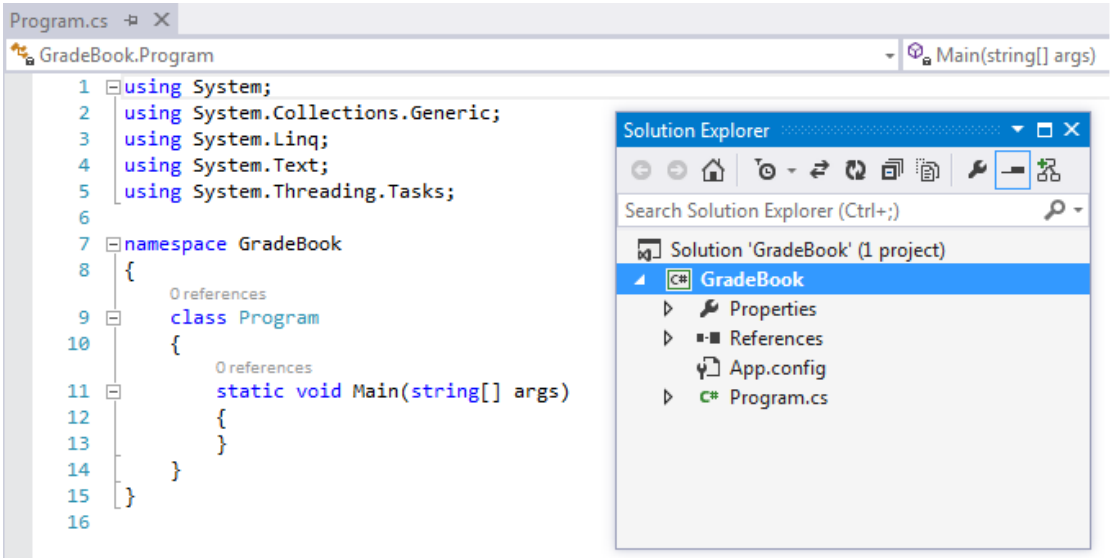
1. افتح محيط العمل Visual Studio 2013.
2. أنشئ مشروع جديد:

File → New → Project

3. اختر Visual C# / Console Application، وقم بإدخال اسم المشروع GradeBook ومسار التخزين:

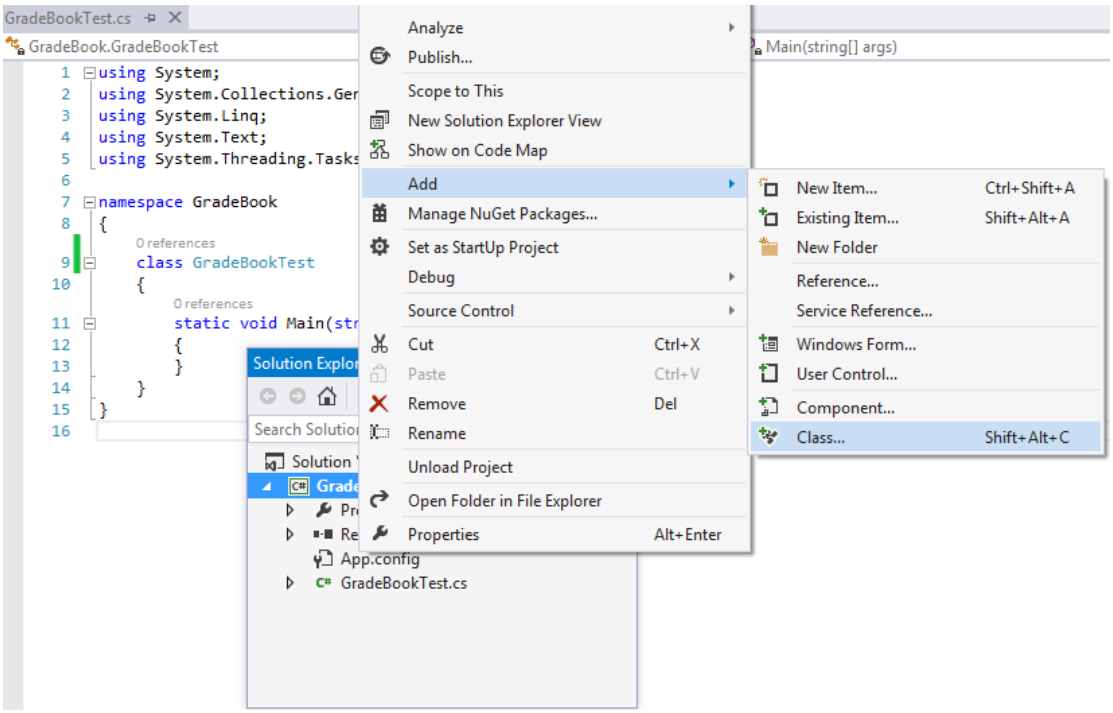


4. يتم فتح مشروع جديد يحوي الملف Program.cs والذي بداخلة الطريقة ( ) Main والتي هي نقطة البدء بالتنفيذ:



5. قم بتغيير اسم الملف Program.cs إلى GradeBookTest.cs. لاحظ أن اسم الصف سيُصبح تلقائياً أيضاً GradeBookTest.

6. انقر بالزر الأيمن على أيقونة المشروع GradeBook ثم اختر إضافة Add صف Class:



7. قم بتسمية الصف الجديد GradeBook.

8. قم بكتابة الطريقة البسيطة `DisplayMessage()` في الصف والتي تُظهر رسالة ترحيبية:

```
// GradeBook.cs
// Class declaration with one method.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public void DisplayMessage()
{
 Console.WriteLine( "Welcome to the Grade Book!" );
} // end method DisplayMessage
} // end class GradeBook
```

9. افتح الملف `GradeBookTest.cs` لإنشاء غرض من الصف واستدعاء طريقة الصف:

```
//  GradeBookTest.cs
// Create a GradeBook object and call its DisplayMessage method.
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
 GradeBook myGradeBook;
 myGradeBook = new GradeBook();
// call myGradeBook's DisplayMessage method
 myGradeBook.DisplayMessage();
} // end Main
} // end class GradeBookTest
```

10. قم بالتنفيذ:

```
Welcome to the Grade Book!
Press any key to continue . . .
```

11. لاحظ أنك بعد إضافتك للصف الجديد `GradeBook` أصبح لديك نمط بيانات جديد يُمكنك تعريف متغيرات منه وإنشاء أغراض جديدة.
12. يتم استخدام المعامل `new` لإنشاء متغير (غرض object، مُنتسخ instance) جديد من الصف `GradeBook`.

لاحظ المعامل (.) بعد اسم المتغير لاستدعاء طريقة الصف.

التصريح عن طريقة المعاملات الفرق بين

التصريح عن طريقة لها معاملات:

- نقوم فيما يلي بإضافة معامل الدخل `courseName` (اسم المادة) لطريقة الصف السابق ليُصبح الصف:

```
// GradeBook.cs
// Class declaration with a method that has a parameter.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public void DisplayMessage( string courseName )
{
 Console.WriteLine( "Welcome to the grade book for\n{0}!", courseName );
} // end method DisplayMessage
} // end class GradeBook
```

- ثم نقوم بإنشاء غرض من الصف السابق واستدعاء الطريقة `DisplayMessage` مع تمرير قيمة لمعامل الدخل. يتم الطلب من المستخدم بإدخال سلسلة نصية (اسم المادة) ومن ثم تمرير القيمة كمعامل دخل للطريقة:

```
// GradeBookTest.cs
// Create a GradeBook object and pass a string to
// its DisplayMessage method.
using System;
public class GradeBookTest
{
// Main method begins program execution
public static void Main( string[] args )
{
// create a GradeBook object and assign it to myGradeBook
 GradeBook myGradeBook = new GradeBook();
// prompt for and input course name
Console.WriteLine( "Please enter the course name:" );
string nameOfCourse = Console.ReadLine(); // read a line of text
Console.WriteLine(); // output a blank line
// call myGradeBook's DisplayMessage method
// and pass nameOfCourse as an argument
myGradeBook.DisplayMessage(nameOfCourse);
} // end Main
} // end class GradeBookTest
```

- يكون التنفيذ مثلاً:

```
Please enter the course name:
programming

Welcome to the grade book for
programming!
Press any key to continue . . .
```

2. الطرق الساكنة

الأهداف التعليمية:

الطرق الساكنة.

المتغيرات والطرق الساكنة:

- تُنفذ لطرق في الكثير من الأحيان على أغراض معينة. إلا أنه في لعديد من لحالات يُمكن أن تهدف الطريقة إلى القيام بتنفيذ مهمة لاتتعلق بمحتوى أي غرض. تُسمى هذه الطرق بالطرق لساكنة ويتم التصريح عنها باستخدام الكلمة المفتاحية static.
- يتم استدعاء طريق ساكنة من صف بكتابة اسم لصف متبوعاً بالمعامل (.) ثم اسم الطريقة مع تمرير معاملاتها إن وجدت بين قوسين:

ClassName MethodName(arguments)

- بالعودة إلى الصف GradeBook المنشئ في المثال السابق. سنقوم بتعديل الطريقة DisplayMessage() بحيث تُصبح ساكنة:

```
// GradeBook.cs
// Class declaration with one method.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public static void DisplayMessage()
{
 Console.WriteLine( "Welcome to the Grade Book!" );
} // end method DisplayMessage
} // end class GradeBook
```

- بالعودة إلى الملف GradeBookTest.cs لاستدعاء طريقة الصف الساكنة السابقة:

```
//  GradeBookTest.cs
public class GradeBookTest
{
   // Main method begins program execution
   public static void Main( string[] args )
   {
       GradeBook.DisplayMessage();

   } // end Main
} // end class GradeBookTest
```

- يكون ناتج التنفيذ:

```
Welcome to the Grade Book!
Press any key to continue . . .
```

التصريح عن طريقة ساكنة لها معاملات:

- نقوم فيما يلي بإضافة معامل الدخل courseName (اسم المادة) لطريقة الصف السابق الساكنة ليُصبح الصف:

```
// GradeBook.cs
// Class declaration with a method that has a parameter.
using System;
public class GradeBook
{
// display a welcome message to the GradeBook user
public static void DisplayMessage( string courseName )
{
 Console.WriteLine( "Welcome to the grade book for\n{0}!", courseName );
} // end method DisplayMessage
} // end class GradeBook
```

- نقوم باستدعاء الطريقة الساكنة السابقة ضمن الطريقة Main. يتم الطلب من المستخدم بإدخال سلسلة نصية (اسم المادة) ومن ثم تمرير القيمة كمعامل دخل للطريقة:

```
using System;
public class GradeBookTest
{
   // Main method begins program execution
   public static void Main( string[] args )
   {
      // prompt for and input course name
      Console.WriteLine( "Please enter the course name:" );
      string nameOfCourse = Console.ReadLine(); // read a line of text
      Console.WriteLine(); // output a blank line
      // call DisplayMessage method
      // and pass nameOfCourse as an argument
      GradeBook.DisplayMessage(nameOfCourse);
   } // end Main
} // end class GradeBookTest
```

- يكون التنفيذ مثلاً:

```
Please enter the course name:
Programming

Welcome to the grade book for
Programming!
Press any key to continue. . .
```

الكلمة المفتاحية public:

- يجب التصريح عن طريقة في صف بأنها عامة كي نستطيع استدعائها في صف آخر.
- يتم التصريح عن طريقة بأنها عامة باستخدام الكلمة المفتاحية public.
- في حال كان استدعاء الطريقة ضمن نفس الصف فليس من الضروري التصريح عن الطريقة بأنها عامة.
- نقوم في المثال التالي بالتصريح عن الطريقة DisplayMessage ضمن نفس الصف GradeBookTest واستدعائها في الطريقة Main للصف نفسه:

```
using System;
public class GradeBookTest
{
     static void DisplayMessage(string courseName)
    {
        Console.WriteLine("Welcome to the grade book for\n{0}!",
           courseName);
    } // end method DisplayMessage

   // Main method begins program execution
   public static void Main( string[] args )
   {
       // prompt for and input course name
       Console.WriteLine("Please enter the course name:");
       string nameOfCourse = Console.ReadLine(); // read a line of text
       Console.WriteLine(); // output a blank line
       // call DisplayMessage method
       // and pass nameOfCourse as an argument
       DisplayMessage(nameOfCourse);
   } // end Main
} // end class GradeBookTest
```

- يكون ناتج التنفيذ:

```
Please enter the course name:
Programming

Welcome to the grade book for
Programming!
Press any key to continue . . .
```

3. الطريقة Main

الأهداف التعليمية:

الطريقة الساكنة Main.

الطريقة الساكنة Main:

- يبدأ التنفيذ من الطريقة الساكنة Main والتي تكون مُعرفة عادةً كما يلي:

```
public static void Main(string[] args)
```

- يُمكنك حذف المعاملات للطريقة Main في حال عدم الحاجة إليها:

```
public static void Main()
```

- عندما تقوم بتنفيذ التطبيق من سطر التعليمات (command line)، يُمكنك كتابة اسم التطبيق متبوعاً بقيم لهذه المعاملات كما يُبين المثال التالي:

```
using System;
public class MathTest
{
    // obtain three floating-point values and determine maximum value
    public static void Main(string[] args)
    {
        Console.WriteLine("Welcome to " + args[0] + " from " + args[1] );
    } // end Main
}
```

- يُمكنك في هذ المثال فتح نافذة الأوامر:

Accessories → Command Prompt

- اكتب اسم الملف التنفيذي متبوعاً بقيم لمعاملات:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\>TestMain.exe C# Microsoft
Welcome to C# from Microsoft

C:\>
```

ملاحظة: قمنا في مثالنا بنسخ الملف التنفيذي إلى المجلد C:\.

4.التمارين

تمرين:

قم بإنشاء مشروع جديد من النمط Console Application. قم بكتابة طريقة تطلب من المستخدم إدخال اسمه ومن ثم تقوم بطباعة رسالة ترحيبية تحوي الاسم المدخل. قم باستدعاء هذه الطريقة ضمن الطريقة Main.

أساسيات C#

الكلمات المفتاحية:

المتغيرات، أنماط البيانات، التحويل الضمني والتحويل الصريح، العمليات الحسابية، العمليات المنطقية، الإسناد، الثوابت، الترقيم، التعليمات الشرطية.

ملخص:

نعرض في هذا الفصل أساسيات البرمجة باستخدام لغة C#.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- المتغيرات
- أنماط البيانات
- التحويل الضمني والتحويل الصريح
- العمليات الأساسية
- التصريح عن الثوابت والتصريح عن الترقيم
- التعليمات الشرطية

المخطط:

يتضمن فصل "أساسيات C#" 3 وحدات عناوينها بالترتيب المحدد:

1. أساسيات C#.
2. الثوابت والترقيم.
3. التعليمات الشرطية.

1. أساسيات C#

الأهداف التعليمية:

المتغيرات، أنماط البيانات، التحويل الضمني والتحويل الصريح، العمليات الحسابية، العمليات المنطقية.

المتغيرات:

- المتغير variable هو عنوان منطقة محجوزة في الذاكرة يُمكن الكتابة فيها والقراءة منها
- يتم التصريح عن المتغيرات باستخدام الشكل العام التالي:

type variablename ;

أمثلة:

```
int m;
int n, k;
int l = 100;
int a = 100, b, c = 200, d;
```

- يُمكن أن يحتوي اسم المتغير على أحرف أو أرقام أو إشارة
- يجب أن يبدأ اسم المتغير بحرف أو إشارة _
- لا يُمكن استخدام كلمات اللغة المفتاحية كإسم للمتغير.
- تتأثر اللغة بحالة الأحرف فالمتحول x هو غير X.

ملاحظة: لا يجوز التعامل مع متحول لم يُعطى قيمة ابتدائية. فالتعليمات التالية مثلاً:

```
int i, j;
i = j;
```

ستولد الخطأ التالي (استخدام متغير دون قيمة):

Use of unassigned local variable j

أنماط البيانات:

يُحدّد نمط بيانات المتغير القيم التي يُمكن أن تُسند للمتغير. كما يختلف حجم التخزين المطلوب للمتغير حسب نمطه.

1. أنماط الأعداد الصحيحة:

نعرض في الجدول التالي جميع أنماط الأعداد الصحيحة التي توفرها لغة C#.

الاسم	مجال القيم	الوصف
sbyte	−128 to 127	Signed 8-bit integer
byte	0 to 255	Unsigned 8-bit integer
short	−32,768 to 32,767	Signed 16-bit integer
ushort	0 to 65,535	Unsigned 16-bit integer
int	−2,147,483,648 to 2,147,483,647	Signed 32-bit integer
uint	0 to 4,294,967,295	Unsigned 32-bit integer
long	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer

2. أنماط الأعداد العشرية:

نعرض في الجدول التالي جميع أنماط الأعداد العشرية التي توفرها C#.

الاسم	مجال القيم	الوصف
float	±1.5 × 10−45 to ±3.4 × 1038	7 digits precision
double	±5.0 × 10−324 to ±1.7 × 10308	15-16 digits precision
decimal	±1.0 × 10−28 to ±7.9 × 1028	28-29 significant digits

3. النمط المنطقي:

يُستخدم هذا النمط لتخزين قيمة منطقية صح أم خطأ.

الاسم	القيم	الوصف
bool	true, false	قيمة منطقية صح أم خطأ

4. النمط المحرفي:

يُستخدم هذا النمط لتخزين محرف واحد.

الاسم	القيم	الوصف
char	one char	يُمكن تخزين محرف واحد

5. السلاسل النصية:

يُستخدم هذا النمط لتخزين سلسلة من المحارف.

الاسم	القيم	الوصف
string	string of chars	يُمكن تخزين سلسلة من المحارف

أمثلة:

```
char a = 'f';
string s1 = "Hello ";
string s2 = "World";
string s3 = s1 + s2;
//s3 is now "Hello World"
```

التحويل الضمني Implicit Conversion:

يُمكن 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;
```

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

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

التحويل الصريح Explicit Conversion:

- يُمكن أيضاً إجراء عملية "قصر" بحيث نقوم بإسناد متغير من نمط ذو مجال قيم أكبر إلى نمط ذو مجال قيم أقل. بالطبع، يُمكن ألا تُعطي هذه العملية نتائج صحيحة في حال كانت القيم المسندة خارج مجال المتغير المسند إليه
- يُمكن استخدام الطريقة checked لإطلاق استثناء في حال كانت عملية القصر لا تُعطي نتائج صحيحة

أمثلة:

```
//Explicit Conversion
long v = 30000;
int i= (int) v ;
//A valid Cast. The max is 2147483647
// i will be 30000
long g=3000000000;
int j= (int) v;
//An invalid Cast. The max is 2147483647
//j will be -129947296
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
```

العمليات الحسابية:

تحوي لغة C# العمليات الحسابية التالية:

1. (+) الجمع:

7 + 4 = 11
"Hello" + "Mamy" = "Hello Mamy"

2. (−) الطرح
3. (*) الضرب
4. (/) القسمة الصحيحة
5. (/) القسمة العشرية
6. (%) باقي القسمة

أمثلة:

```
int n = 10;
int m = 3;
int r = n / m;
//  r will be 3

double n = 10;
double m = 3;
double r = n / m;
// r will be 3.333

char c = 'A';
int i;
i = c + 1;
// i will be 66
```

العمليات المنطقية:

تحوي لغة C# العمليات المنطقية التالية:

1. (&&) عملية و AND
2. (||) عملية أو OR
3. (!) عملية النفي

أمثلة:

```
bool b;
int n = 10;
int m = 20;
b = ((n > m) || (m % n == 0));
// b will be true
```

معاملات المقارنة:

تحوي لغة C# معاملات المقارنة التالية:

1. (==) يساوي
2. (=!) لا يساوي
3. (>) أكبر
4. (<) أصغر
5. (=>) أكبر أو يساوي
6. (=<) أصغر أو يساوي

معاملات الزيادة والانقاص:

تحوي لغة C# المعاملات التالية:

1. (++) زيادة واحد
2. (--) انقاص واحد

أمثلة:

```
int i = 10;
i++;          //(i= i +1)   i will be 11
i--;        //(i= i -1)    i will be 10
```

الإسناد:

- يُمكن استخدام الإسناد كما يلي:

1. (=) الإسناد
2. (+=) الجمع ثم الإسناد
3. (-=) الطرح ثم الإسناد
4. (*=) الضرب ثم الإسناد
5. (/=) القسمة ثم الإسناد
6. (%=) باقي القسمة ثم الإسناد

أمثلة:

```
int i = 10;
i += 5;       // (i = i + 5)       i will be 15
i -= 10;     //  (i = i - 10 )     i will be 5
```

معامل الاختبار الثلاثي: ? The Ternary Operator:

- يكون لهذا المعامل الشكل التالي:

condition ? true-value : false-value

- يُستخدم لاختبار شرط معين وإعادة قيمة إذا كان الشرط محققاً، وإعادة قيمة أخرى في حال عدم تحقق الشرط.
- يُبين المثال التالي استخدام هذا المعامل:

```
int x = 1;
string s = x.ToString() + " ";
s = s + (x == 1 ? "man" : "men");
```

2. الثوابت والترقيم

الأهداف التعليمية:

الثوابت، الترقيم.

الثوابت:

يُمكن التصريح عن قيم ثابتة باستخدام لشكل العام التالي:

const *Type constname = expression*

لا يُمكن تعديل قيمة الثابت بعد التصريح عنه.

مثال

```
const double pi = 3.14;
double r, x;
r = 20;
x = 2 * pi * r;
```

الترقيم:

- يُمكن تعريف نمط جديد يتألف من قائمة من الثوابت وذلك باستخدام الكلمة المفتاحية enum
- تكون القيمة الافتراضية للعنصر الأول في قائمة الثوابت 0 وللعنصر الثاني القيمة 1 وهكذا....

مثال:

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

- يُمكن إسناد قيم جديدة عوضاً عن القيم الافترضية كما في لمثال لتالي:

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

حيث ستبدأ قيم الثوابت اعتباراً من 1 بدلاً من 0.

- يكون النمط الافتراضي لثوابت الترقيم int. كما يُمكن اختيار نمط من الأنماط التالية

byte, sbyte, short, ushort, int, uint, long, ulong

كما يُبين المثال التالي:

```
enum Days : byte {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

- يُمكن استخدام النمط المُعرّف بالترقيم للتصريح عن متغيرات وإسناد قيم لها كما يُبين المثال التالي:

```
using System;
public class EnumTest
{
    enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    static void Main()
    {
        Days x, y;
        x = Days.Sat;
        y = Days.Mon;
        int z = x - y;
        Console.WriteLine(z);
    }
}
```

- يكون ناتج التنفيذ:

```
5
Press any key to continue . . .
```

- كما يُمكن إجراء عملية قصر للحصول على قيمة ثابت. مثلاً:

```
public class EnumTest
{
    enum Days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };

    static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Fri;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Fri = {0}", y);
    }
}
```

- يكون ناتج التنفيذ:

```
Sun = 0
Fri = 5
Press any key to continue . . .
```

- نقوم في المثال التالي بالتصريح عن ترقيم من النمط long ومن ثم إجراء عملية قصر للحصول على قيم العناصر:

```
public class EnumTest2
{
    enum Range : long { Max = 2147483648L, Min = 255L };
    static void Main()
    {
        long x = (long)Range.Max;
        long y = (long)Range.Min;
        Console.WriteLine("Max = {0}", x);
        Console.WriteLine("Min = {0}", y);
    }
}
```

- يكون ناتج التنفيذ:

```
Max = 2147483648
Min = 255
Press any key to continue . . .
```

3. التعليمات الشرطية

الأهداف التعليمية:

التعليمات الشرطية.

التعليمة الشرطية if else:

- يكون للتعليمة الشرطية if else الشكل العام التالي:

```
if (condition)
{ statement (s)-if-true }
else
{ statement (s)-if-false }
```

حيث:

1. *condition* تعبير منطقي يُعيد true أو false
2. *statement (s) if true* التعليمات التي تُنفذ إذا تحقق الشرط.
3. *statement (s) if false* التعليمات التي تُنفذ إذا لم يتحقق الشرط

- يُمكن للتعليمة لشرطية if ألا تحوي على else:

```
if (condition)
{
  // This executes if
 // condition==true
}
```

- كما يُمكن للتعليمة الشرطية if أن تحوي على عدة شروط:

```
if (condition1)
{
//This executes if condition1==true
}
else if (condition2)
{
//This executes if
//condition1==false and condition2==true
}
else
{
//This executes if neither
// condition1 nor condition2 are true
}
```

أمثلة:

مثال1:

```
int Number;
int Digits;
Number = 88;
if (Number < 10)
            Digits = 1;
else if (Number < 100)
            Digits = 2;
else if (Number < 1000)
            Digits = 3;
else
         Digits = 4;
```

مثال2:

```
char gender='x';
. . .
if (gender=='m' || gender=='f')
{
  // it is a male or female
}
if (gender !='m' && gender != 'f')
{
  // neither a male nor a female
}
```

مثال3:

```
char gender='x';
. . .
If (gender=='m' || gender=='f')
{
  // it is a male or female
}
else
{
  // neither a male nor a female
}
```

مثال4:

```
char gender='x';
. . .
if (gender=='m')
{
       // it is a male
}
else
```

```
{
	if (gender =='f')
	{
		// it is a female
	}
	else
	{
		// neither a male  nor a female
	}
}
```

مثال5:

```
char gender='x';
. . .
if (gender=='m')
{
	// it is a male
}
else If (gender =='f')
{
	// it is a female
}
else
{
	// neither a male nor a female
}
```

مثال6:

```
if (roll==1)
	// roll is 1
else If (roll ==2 )
	// roll is 2
else If (roll ==3 )
	// roll is 3
else If (roll ==4 )
	// roll is 4
else If (roll ==5 )
	// roll is 5
else If (roll ==6 )
	// roll is 6
else
// it isn't a number from 1 to 6
```

التعليمة الشرطية switch:

يكون للتعليمة الشكل العام التالي:

```
switch (value)
{
case result_1:
                    // do stuff for result_1
                    break;
case result_2:
                    // do stuff for result_2
                    break;
. . .
case result_n:
                    // do stuff for result n
                    break;
default:
                    // do stuff for result case
                    break;
}
```

أمثلة:

مثال1:

```
int i;
. . .
switch (i)
{
case 1:
              // i is 1
              break;
case 2:
              // i as 2
              break;
case 3:
              // i is 3
              break;
default:
              // i is not 1,2 or 3";
              break;
}
```

مثال2:

```
switch (roll)
        {
            case 1:
                //roll is 1
                break;
            case 2:
                //roll is 2
                break;
            case 3:
                //roll is 3
                break;
            case 4:
                //roll is 4
                break;
            case 5:
                //roll is 5
                break;
            case 6:
                //roll is 6
                break;
            default:
                //it isn't a number from 1 to 6
                break;
        }
```

مثال3:

```
switch (roll)
        {
            case 1:
            case 3:
            case 5:
                //roll is odd
                break;
            case 2:
            case 4:
            case 6:
                //roll is even
                break;
            default:
                //it isn't a number from 1 to 6
                break;
        }
```

مثال4:

```
string Country, Language;
. . . .
Switch (Country)
{
case "au":
case "uk":
case "us":
              Language="English";
              break;
case "at":
case "de":
              Language="German";
              break;
}
```

4. التمارين

تمرين 1:

ماهي الأخطاء التي ستظهر عند ترجمة التعليمات التالية؟

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

تمرين 2:

ماذا ستكون قيمة المتغير m في نهاية 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;
```

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

```
int i = 12;
int j = 5;
int m;
m = (i % j == 0) ? i : j;
```

```
int i = 12;
int j = 5;
int m;
m = (i / j == 0) ? i : j;
```

```
double i = 10.5;
double j = 5.5;
double m;
m = i / j * j + i % j;
```

تمرين3:

قم بكتابة تطبيق لحل معادلة من الدرجة الثانية:

$$A\,x^2 + B\,x + C = 0$$

يقوم التطبيق بسؤال المستخدم لإدخال القيم A, B, C.

بعدها يُظهر التطبيق جذور المعادلة إن وجدت أو عبارة "لاجذور حقيقية".

تذكرة:

لحساب جذور المعادلة:

$$A\,x^2 + B\,x + C = 0$$

نحسب:

$$D = B^2 - 4*A*C$$

- في حال كان D==0 فللمعادلة جذر مضاعف هو: –B/2*A
- في حال كان D>0 فللمعادلة جذرين:

x1= –B-√D /2*A
x2= –B+√D /2*A

- في حال كان D<0: فلا يوجد جذور حقيقية للمعادلة.

تعليمات التكرار

الكلمات المفتاحية:

for، while، break، continue.

ملخص:

نعرض في هذا الفصل لمختلف أشكال تعليمات التكرار التي توفرها لغة البرمجة C#.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- تعليمة التكرار for
- تعليمة التكرار while
- تعليمة كسر التكرار break
- تعليمة إيقاف الحلقة continue

المخطط:

يتضمن فصل "تعليمات التكرار" وحدتين عناوينها بالترتيب المحدد:

1. تعليمة التكرار for.
2. تعليمة التكرار while.

1. تعليمة التكرار for

الأهداف التعليمية:

تعليمة التكرار for.

تعليمة التكرار for:

يكون لتعليمة التكرار for الشكل العام التالي:

```
for (initializer; condition; iterator)
    statement (s)
```

حيث:

1. *initializer*: التعليمات الابتدائية التي تُنفذ أولاً.
2. *condition*: يستمر التكرار (تنفيذ تعليمات الحلقة) طالما أن هذا الشرط محقق.
3. *iterator*: تعليمات تُنفذ بعد الانتهاء من كل تنفيذ للتعليمات داخل الحلقة.

مثال:

يُبين لمثال التالي استخدام حلقة for لطباعة الأعداد من 1 إلى 10:

```
using System;
public class ForCounter
{
   public static void Main( string[] args )
   {
      // for statement header includes initialization,
      // loop continuation condition and increment
      for ( int counter = 1; counter <= 10; counter++ )
         Console.Write( "{0}  ", counter );

      Console.WriteLine(); // output a newline
   } // end Main
} // end class ForCounter
```

- يكون ناتج التنفيذ:

```
1 2 3 4 5 6 7 8 9 10
Press any key to continue . . .
```

مثال:

- نقوم في المثال التالي بإيجاد مجموع الأعداد الزوجية المحصورة بين 2 و20:

```
// Sum.cs
// Summing integers with the for statement.
using System;
public class Sum
{
   public static void Main( string[] args )
   {
      int total = 0; // initialize total

      // total even integers from 2 through 20
      for ( int number = 2; number <= 20; number += 2 )
         total += number;

      Console.WriteLine( "Sum is {0}", total ); // display results
   } // end Main
} // end class Sum
```

- يكون ناتج التنفيذ:

```
Sum is 110
Press any key to continue . . .
```

مثال:

- نقوم في المثال التالي بإظهار المبلغ الناتج في نهاية كل سنة بعد حساب الفائدة المتوجبة:

```
// Interest.cs
// Compound-interest calculations with for.
using System;

public class Interest
{
   public static void Main( string[] args )
   {
      decimal amount; // amount on deposit at end of each year
      decimal principal = 1000; // initial amount before interest
      double rate = 0.05; // interest rate

      // display headers
      Console.WriteLine( "Year{0,20}", "Amount on deposit" );

      // calculate amount on deposit for each of ten years
      for ( int year = 1; year <= 10; ++year )
      {
         // calculate new amount for specified year
         amount = principal *
            ( ( decimal ) Math.Pow( 1.0 + rate, year ) );

         // display the year and the amount
         Console.WriteLine( "{0,4}{1,20:C}", year, amount );
      } // end for
   } // end Main
} // end class Interest
```

- يكون ناتج التنفيذ:

```
Year  Amount on deposit
  1         $1,050.00
  2         $1,102.50
  3         $1,157.63
  4         $1,215.51
  5         $1,276.28
  6         $1,340.10
  7         $1,407.10
  8         $1,477.46
  9         $1,551.33
 10         $1,628.89
Press any key to continue . . .
```

تعليمة كسر الحلقة break:

تقوم هذه التعليمة بكسر التكرار بمعنى الخروج من الحلقة.

مثال:

يُبين المثال التالي استخدام التعليمة break حيث يتم كسر الحلقة والخروج منها عند تنفيذ هذه التعليمة:

```
// BreakTest.cs
// break statement exiting a for statement.
using System;

public class BreakTest
{
   public static void Main( string[] args )
   {
      int count; // control variable also used after loop terminates

      for ( count = 1; count <= 10; ++count ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            break; // terminate loop

         Console.Write( "{0} ", count );
      } // end for

      Console.WriteLine( "\nBroke out of loop at count = {0}", count );
   } // end Main
} // end class BreakTest
```

- يكون ناتج التنفيذ:

```
1 2 3 4
Broke out of loop at count = 5
Press any key to continue . . .
```

تعليمة إيقاف الحلقة continue:

تقوم هذه التعليمة بإيقاف تنفيذ التعليمات المتبقية بعدها في التكرار الحالي والانتقال إلى التكرار التالي.

مثال:

- يُبين المثال التالي استخدام التعليمة continue حيث يتم تخطي تعليمات التكرار الحالي عند تنفيذها والانتقال إلى التكرار التالي:

```
// ContinueTest.cs
// continue statement terminating an iteration of a for statement.
using System;

public class ContinueTest
{
   public static void Main( string[] args )
   {
      for ( int count = 1; count <= 10; ++count ) // loop 10 times
      {
         if ( count == 5 ) // if count is 5,
            continue; // skip remaining code in loop

         Console.Write( "{0} ", count );
      } // end for

      Console.WriteLine( "\nUsed continue to skip displaying 5" );
   } // end Main
} // end class ContinueTest
```

- يكون ناتج التنفيذ:

```
1 2 3 4 6 7 8 9 10
Used continue to skip displaying 5
Press any key to continue . . .
```

2. تعليمة التكرار while

الأهداف التعليمية:

تعليمة التكرار while.

تعليمة التكرار while:

يكون لتعليمة التكرار while الشكل العام التالي:

While (*condition*)
statement

يتم اختبار الشرط *condition* وتنفيذ التعليمات *statement (s)* إذا كان الشرط محقق. يستمر تكرار تنفيذ التعليمات طالما أن الشرط محقق.

مثال:

- يُبين لمثال لتلي استخدم تعليمة التكرار while لطباعة الأعداد من 1 إلى 10:

```
// WhileTest.cs
using System;
public class DoWhileTest
{
   public static void Main( string[] args )
   {
      int counter = 1; // initialize counter
      while (counter <= 10)
      {
         Console.Write( "{0}  ", counter );
         counter++;
      }
      Console.WriteLine(); // outputs a newline
   } // end Main
} // end class WhileTest
```

- يكون ناتج التنفيذ:

```
1 2 3 4 5 6 7 8 9 10
Press any key to continue
```

تعليمة التكرار do while:

يكون لتعليمة التكرار do while الشكل العام التالي:

```
Do
        Statement (S)
While (condition)
```

يتم تنفيذ التعليمات *statement (s)* ثم اختبار الشرط *condition*. يتم تكرار التنفيذ طالما الشرط محقق.

مثال:

- يُبين المثال التالي استخدام التعليمة do while لطباعة الأعداد من 1 إلى 10:

```
// DoWhileTest.cs
// do...while repetition statement.
using System;

public class DoWhileTest
{
   public static void Main( string[] args )
   {
      int counter = 1; // initialize counter

      do
      {
         Console.Write( "{0}  ", counter );
         ++counter;
      } while ( counter <= 10 ); // end do...while

      Console.WriteLine(); // outputs a newline
   } // end Main
} // end class DoWhileTest
```

- يكون ناتج التنفيذ:

```
1 2 3 4 5 6 7 8 9 10
Press any key to continue . . .
```

تعليمة كسر الحلقة break:

تقوم هذه التعليمة بكسر التكرار بمعنى الخروج من الحلقة.

مثال:

- يُبين المثال التالي استخدام التعليمة break حيث يتم كسر الحلقة والخروج منها عند تنفيذ هذه التعليمة:

```
// BreakTest.cs
// break statement exiting a for statement.
using System;

public class BreakTest
{
   public static void Main( string[] args )
   {
      int count=0; // control variable also used after loop terminates

      while ( count <= 10 ) // loop 10 times
      {
          count++;
         if ( count == 5 ) // if count is 5,
            break; // terminate loop

          Console.Write( "{0} ", count );
      } // end while

      Console.WriteLine( "\nBroke out of loop at count = {0}", count );
   } // end Main
} // end class BreakTest
```

- يكون ناتج التنفيذ:

```
1 2 3 4
Broke out of loop at count = 5
Press any key to continue . . .
```

تعليمة إيقاف الحلقة continue:

تقوم هذه التعليمة بإيقاف تنفيذ التعليمات المتبقية بعدها في التكرار الحالي والانتقال إلى التكرار التالي.

مثال:

- يُبين المثال التالي استخدام التعليمة continue حيث يتم تخطي تعليمات التكرار الحالي عند تنفيذها والانتقال إلى التكرار التالي:

```
// ContinueTest.cs
// continue statement terminating an iteration of a for statement.
using System;

public class ContinueTest
{
   public static void Main( string[] args )
   {
       int count = 0;
      while ( count <= 10 ) // loop 10 times
      {
          count++;
          if ( count == 5 ) // if count is 5,
            continue; // skip remaining code in loop

         Console.Write( "{0} ", count );
      } // end for

      Console.WriteLine( "\nUsed continue to skip displaying 5" );
   } // end Main
} // end class ContinueTest
```

- يكون ناتج التنفيذ:

```
1 2 3 4 6 7 8 9 10
Used continue to skip displaying 5
Press any key to continue . . .
```

3. التمارين

تمرين1:

ما قيمة المتغير s بعد تنفيذ كل من التعليمات التالية:

أ. الحالة 1

```
int s = 0;
for (int i = 1; i <= 5; i++)
{
  s = s + i;
}
```

ب. الحالة 2

```
int s = 1;
for (int i = 1; i <= 5; i++)
{
  s = s * i;
}
```

ج. الحالة 3

```
int s = 0;
for (int i = 1; i <= 5; i++)
{
 if (i % 2 == 1)
    s = s + i;
}
```

د. الحالة 4

```
int s = 0;
for (int i = 1; i <= 5; i = i + 2)
{
   s = s + i;
}
```

هـ. الحالة 5

```
int s = 0, i = 1;
while (i <= 5)
{
   s = s + i;
   i = i + 1;
}
```

و. الحالة 6

```
int s = 0, i = 1;
while (i <= 5)
{
   i = i + 1;
   s = s + i;
}
```

ز. الحالة 7

```
int s = 0, i = 0;
while (i <= 5)
{
  i = i + 1;
  if (i % 2 == 0) break;
       s = s + i;
}
```

ح. الحالة 8

```
int s = 0, i = 0;
while (i <= 5)
{
   i = i + 1;
   if (i % 2 == 0) continue;
   s = s + i;
}
```

ط. الحالة 9

```
int s = 0, i = 0;
do
{
  i = i + 1;
  s = s + i;
}
while (i <= 5);
```

ي. الحالة 10

```
int s = 0, i = 0;
do
{
  s = s + i;
  i = i + 1;
}
while (i <= 5);
```

الحالات	الإجابة الصحيحة
الحالة 1	S=15
الحالة 2	S=120
الحالة 3	S=9
الحالة 4	S=9
الحالة 5	S=15
الحالة 6	S=20
الحالة 7	S=1
الحالة 8	S=9
الحالة 9	S=21
الحالة 10	S=15

تمرين2:

قم بكتابة تطبيق يطلب من المستخدم إدخال عدد صحيح. يقوم التطبيق باختبار فيما إذا كان العدد أولي أم لا ويُظهر رسالة موافقة.

تمرين3:

قم بكتابة تطبيق يطلب من المستخدم إدخال عددين صحيحين. يقوم التطبيق بحساب القاسم المشترك الأعظم للعددين ومن ثم طباعته.

ملاحظة: يُمكنك استخدام الخوارزمية التالية لحساب القاسم المشترك الأعظم: كرر طرح العدد الكبير من العدد الصحيح حتى يُصبح العددين متساويين.

تمرين4:

قم بكتابة تطبيق يقوم بطباعة الشكل التالي:

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

تمرين5:

قم بكتابة تطبيق يقوم بطباعة الشكل التالي:

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

تمرين6:

قم بكتابة تطبيق يقوم بطباعة الشكل التالي:

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

تمرين7:

قم بكتابة تطبيق يقوم بطباعة الشكل التالي:

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

تمرين8:

قم بكتابة تطبيق يقوم بطباعة الشكل التالي:

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

أساسيات الطرق

الكلمات المفتاحية:

الطرق، طرق الصف Math، معاملات الطرق.

ملخص:

نُبين أولاً أهمية استخدام الطرق في تجزئة البرامج. نستعرض بعدها أهم طرق الصف Math. نختم الفصل بموضوع معاملات الطرق.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- استخدام الطرق
- طرق الصف Math
- معاملات الطرق

المخطط:

يتضمن فصل "أساسيات الطرق" 3 وحدات عناوينها بالترتيب المحدد:

1. تجزئة البرامج باستخدام الطرق.
2. طرق الصف Math.
3. معاملات الطرق.

1. تجزئة البرامج باستخدام الطرق

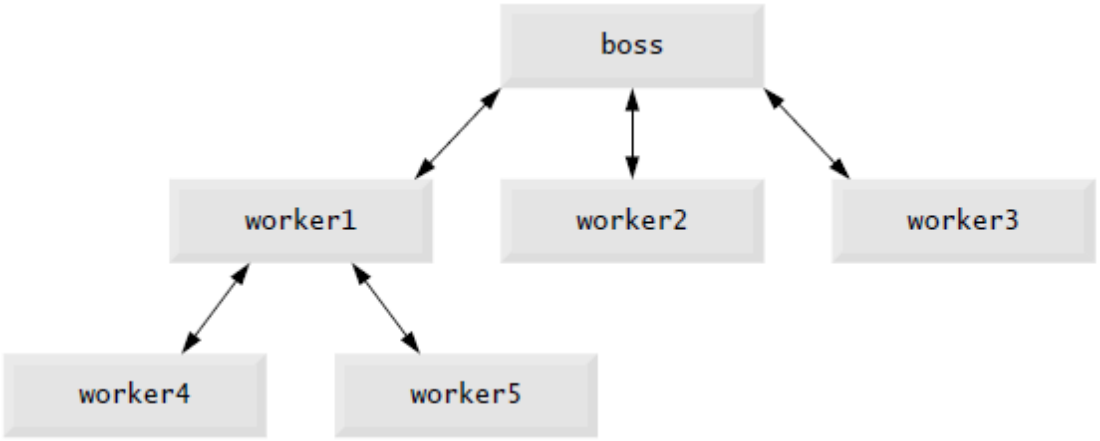
الأهداف التعليمية:

استخدام الطرق لتجزئة البرامج.

- تسمح الطرق Methods (تُدعى أحياناً وظائف functions أو اجرائيات procedures) بتجزئة البرامج إلى مجموعة من الوحدات. تقوم كل وحدة بإنجاز مهمة معينة.
- يُمكنك التصريح عن الطرق في التطبيقات التي تكتبها. تُدعى هذه الطرق عادةً بالطرق المصرح عنها من قبل المستخدم user-defined methods.
- تُكتب تعليمات الطريقة مرة واحدة فقط ويُمكن استخدامها مرات عديدة في التطبيق. تكون تعليمات الطريقة "مخفية" عن الطرق الأخرى.
- يُعتبر مبدأ تجزئة البرنامج إلى وحدات منفصلة من أساسيات هندسة البرمجيات والذي يُساعد في صيانة البرمجيات وإعادة استخدام الوحدات في عدة تطبيقات.

استدعاء الطرق Calling Methods :

- يتم عادةً استدعاء طريقة من قبل طريقة أخرى. عندما ينتهي تنفيذ تعليمات الطريقة المُستدعاة، يعود التحكم إلى الطريقة المُستدعية. يُمكن بالطبع أن تقوم طريقة مُستدعاة باستدعاء طرق أخرى وهكذا....
- يُمكن تشبيه هرمية استدعاء الطرق بطلب مدير (المُستدعي) من عامل (المُستدعى) أن يقوم بتنفيذ مهمة ما وبأن يُعيد تقرير بالنتائج عند الانتهاء من عمله. يُمكن لهذا العامل (وبدون أن يكون المدير على صلة) أن يقوم بطلب تنفيذ مهام من عاملين آخرين.
- يسمح مبدأ إخفاء تفاصيل التنفيذ في البرمجة من تحسين هندسة البرمجيات.
- يُبين الشكل التالي مثلاً هرمية العمل حيث يتواصل المدير boss والتي مع عدة عاملين بشكل هرمي. لاحظ أن العامل worker1 يلعب دور المدير بالنسبة للعاملين worker4 و worker5.



2. طرق الصف Math

الأهداف التعليمية:

طرق الصف Math.

- يحوي الصف Math (الموجود ضمن فضاء الأسماء System) مجموعة من الطرق الرياضية. يُمكن مثلاً حساب الجذر التربيعي لعدد بكتابة:

Math.Sqrt(900.0)

- لاحظ أننا في التعليمة السابقة لم نقم بإنشاء أي غرض من الصف Math
- تأخذ الطريقة Sqrt معامل من النمط double وتُعيد نتيجة من النمط double
- يُمكنك في تطبيق من النوع console كتابة:

Console.WriteLine(Math.Sqrt(900.0));

لإظهار النتيجة (30)

- لاحظ أنه في التعليمة السابقة تُصبح القيمة المُعادة من الطريقة Sqrt هي القيمة الممررة للطريقة WriteLine

- يُمكن أن يكون معامل الطريقة ثابت أو متغير أو تعبير كما يُبين المثال التالي:

مثال مع تنفيذه

```
using System;
public class MathTest
{
    // obtain three floating-point values and determine maximum value
    public static void Main(string[] args)
    {
        const double C = 900.0;
        double x = 16.0;
        double c = 13.0, d = 3.0, f = 4.0;
        Console.WriteLine(Math.Sqrt(C));
        Console.WriteLine(Math.Sqrt(x));
        Console.WriteLine(Math.Sqrt(c + d * f));
    } // end Main
}
```

- يكون الناتج:

```
30
4
5
Press any key to continue . . .
```

طرق الصف Math:

- يحوي الصف Math مجموعة من الطرق الساكنة للعمليات الرياضية.
- كما يحوي الصف Math الثوابت العامة والساكنة:

1. Math.PI = 3.1415926535…
2. Math.E = 2.7182818285…

- يُبين الجدول التالي أهم طرق الصف Math:

الطريقة	الوصف	أمثلة

`Abs( x )`	القيمة المطلقة لـ x	`Abs( 23.7 )` is `23.7`
		`Abs( 0 )` is `0`
		`Abs( -23.7 )` is `23.7`

`Ceiling( x )`	التقريب لأصغر عدد طبيعي ليس أصغر من x	`Ceiling( 9.2 )` is `10.0` `Ceiling( -9.8 )` is `-9.0`
`Cos( x )`	تجيب x (x بالراديان) (x in radians)	`Cos( 0.0 )` is `1.0`
`Exp( x )`	الرفع لقوة العدد e	`Exp( 1.0 )` is approximately `2.7182818284590451` `Exp( 2.0 )` is approximately `7.3890560989306504`
`Floor( x )`	التقريب لأكبر عدد طبيعي ليس أكبر من x	`Floor( 9.2 )` is `9.0` `Floor( -9.8 )` is `-10.0`
`Log( x )`	اللوغاريتم الطبيعي لـ x (القاعدة e)	`Log( 2.7182818284590451 )` is approximately `1.0` `Log( 7.3890560989306504 )` is approximately `2.0`
`Max( x, y )`	أكبر قيمة	`Max( 2.3, 12.7 )` is `12.7` `Max( -2.3, -12.7 )` is `-2.3`
`Min( x, y )`	أصغر قيمة	`Min( 2.3, 12.7 )` is `2.3` `Min( -2.3, -12.7 )` is `-12.7`
`Pow( x, y )`	x مرفوع للقوة y	`Pow( 2.0, 7.0 )` is `128.0` `Pow( 9.0, .5 )` is `3.0`
`Sin( x )`	جيب x (x بالراديان) (x in radians)	`Sin( 0.0 )` is `0.0`
`Sqrt( x )`	الجذر التربيعي لـ x	`Sqrt( 900.0 )` is `30.0` `Sqrt( 9.0 )` is `3.0`

3. معاملات الطرق

الأهداف التعليمية:

معاملات الطرق.

- نقوم فيما يلي بالتصريح عن الطريقة Maximum والتي لها ثلاثة معاملات من النمط double. تقوم هذه الطريقة بإيجاد وإرجاع أكبر قيمة من بين هذه المعاملات الثلاثة.
- يتم في الطريقة Main استدعاء الطريقة السابقة مع تمرير ثلاثة قيم لمعاملاتها.

```
// MaximumFinder.cs
// User defined method Maximum.
using System;
public class MaximumFinder
{
   // obtain three floating point values and determine maximum value
   public static void Main( string[] args )
   {
      // prompt for and input three floating point values
      Console.WriteLine( "Enter three floating point values,\n" +
         "  pressing 'Enter' after each one: " );
      double number1 = Convert.ToDouble( Console.ReadLine() );
      double number2 = Convert.ToDouble( Console.ReadLine() );
      double number3 = Convert.ToDouble( Console.ReadLine() );

      // determine the maximum value
      double result = Maximum( number1, number2, number3 );

      // display maximum value
      Console.WriteLine( "Maximum is: " + result );
   } // end Main

   // returns the maximum of its three double parameters
   public static double Maximum( double x, double y, double z )
   {
      double maximumValue = x; // assume x is the largest to start

      // determine whether y is greater than maximumValue
      if ( y > maximumValue )
         maximumValue = y;

      // determine whether z is greater than maximumValue
      if ( z > maximumValue )
         maximumValue = z;

      return maximumValue;
   } // end method Maximum
} // end class MaximumFinder
```

- يكون ناتج التنفيذ:

```
Enter three floating point values,
  pressing 'Enter' after each one:
2.22
3.33
1.11
Maximum is: 3.33
Press any key to continue . . .
```

4. التمارين

تمرين:

قم بالتصريح عن الطريقة Average والتي تأخذ ثلاثة معاملات من النمط double وتُعيد الوسطي الحسابي لهذه المعاملات الثلاثة.

قم في الطريقة Main بالطلب من المستخدم إدخال ثلاثة أعداد ومن ثم استدعاء الطريقة Average السابقة وإظهار الوسطي الحسابي للأعداد المدخلة.

استخدام الطرق

الكلمات المفتاحية:

استخدام الطرق، معاملات الطرق، القيمة المعادة.

ملخص:

نستعرض في هذا الفصل أساسيات استخدام الطرق وأهم المسائل التي يجب الانتباه إليها.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- أساسيات استخدام الطرق.
- أمثلة عن الطرق.

المخطط:

يتضمن فصل "استخدام الطرق" وحدتين عناوينها بالترتيب المحدد:

1. استخدام الطرق.
2. أمثلة.

1. استخدام الطرق

الأهداف التعليمية:

نستخدم لطرق عادةً بهدف:

1. تجزئة البرنامج إلى أقسام منفصلة، يقوم كل منها بإنجاز عمل معين.
2. تجنب تكرار نفس التعليمات للقيام بأعمال متماثلة.

مسألة رياضية: ليكن المطلوب حساب التوافيق (Combination) لعددين n و m.(عدد المجموعات الجزئية التي تحوي m عنصر والتي يتم اختيارها من n عنصر). يتم حساب التوافيق باستخدام القانون الرياضي التالي:

$$C(n,m) = n! / (m! * (n-m)!)$$

(ترمز الإشارة ! إلى حساب العاملي لعدد: $n! = n*(n-1)*(n-2)* \dots * 1$).

- يُمكن أولاً كتابة البرنامج التالي والذي يقوم بالطلب من المستخدم بإدخال عددين ثم يقوم بحساب التوافيق لهما:

```
using System;
public class Methods
{
   public static void Main( string[] args )
   {
       int N, M, i;

       // prompt for and input two integer values
       Console.WriteLine("Enter two values,\n" +
          "  pressing 'Enter' after each one: ");
       N = Convert.ToInt32(Console.ReadLine());
       M = Convert.ToInt32(Console.ReadLine());

       long factN, factM, factNM, C;

       factN = 1;
       for (i = 1; i <= N; i++)
           factN = factN * i;

       factM = 1;
       for (i = 1; i <= M; i++)
           factM = factM * i;

       factNM = 1;
       for (i = 1; i <= (N   M); i++)
           factNM = factNM * i;

       C = factN / (factM * factNM);
       // display result
       Console.WriteLine("C is: " + C);
   } // end Main
}
```

- يكون ناتج التنفيذ:

```
Enter two values,
  pressing 'Enter' after each one:
5
3
C is: 10
Press any key to continue . . .
```

- لاحظ أن حساب العاملي يُكرر ثلاثة مرات في البرنامج السابق (باستخدام حلقة for)
- لتجاوز مشكلة التكرار نقوم فيما يلي بالتصريح عن الطريقة Fact والتي لها معامل دخل عدد صحيح n وتُعيد العاملي لهذا العدد (عدد صحيح طويل)
- لاحظ أن هذه الطريقة تُستدعى ثلاث مرات في الطريقة Main لحساب التوافيق للعددين N و M.

```
using System;
public class Methods
{
    public static long Fact(int n)
    {
        long f = 1;
        for (int i = 1; i <= n; i++)
            f = f * i;
        return (f);
    }
   public static void Main( string[] args )
   {
       int N, M, i;
       long  C;
       // prompt for and input three floating-point values
       Console.WriteLine("Enter two values,\n" +
          "  pressing 'Enter' after each one: ");
       N = Convert.ToInt32(Console.ReadLine());
       M = Convert.ToInt32(Console.ReadLine());

       C =  Fact(N) / (Fact(M) * Fact(N - M));

       // display result
       Console.WriteLine("C is: " + C);
   } // end Main
}
```

- يكون ناتج التنفيذ:

```
Enter two values,
  pressing 'Enter' after each one:
5
3
C is: 10
Press any key to continue . . .
```

أساسيات استخدام الطرق:

نُبين فيما يلي أساسيات استخدام الطرق:

1. تطابق المعاملات Parameters Matching:

يجب عند استدعاء طريقة الانتباه إلى:

1. تطابق عدد المعاملات الممرره إلى الطريقة.
2. تطابق نمط بيانات كل معامل للطريقة مع القيمة الممرره.

2. الطرق بدون قيمة مرجعة void:

يُمكن أن لا يكون للتابع قيمة مرجعة وعندها نستخدم الكلمة المفتاحية void كما لا نُسند قيمة إرجاع عند الاستدعاء

3. إرجاع قيمة في جميع الحالات:

في حال كانت للطريقة قيمة مُرجعة، فيجب أن تكون هنالك قيمة مرجعة في جميع الحالات. يُعطي المترجم للطريقة التالية:

```
public static bool TestEven(int n)
{
  if (n % 2 == 0)
      return (true);
}
```

الخطأ التالي:

not all code paths return a value

(لا تُعيد جميع المسارات في الطريقة قيمة).

4. توافق نمط القيمة المرجعة:

- يجب توافق نوع المتحول المسند إليه قيمة الطريقة عند الاستدعاء مع نوع القيمة المرجعة
- يُعطي المترجم في المثال التالي:

```
public static bool TestEven(int n)
{
  if (n % 2 == 0)
      return (true);
  else
      return (false);
}
public static void Main( string[] args )
{

  int n;
  n = TestEven(7);
}
```

الخطأ التالي:

Cannot implicitly convert type 'bool' to 'int'

(لا يُمكن التحويل الضمني من النمط bool إلى int).

5. المعاملات الممرره بالقيمة:

يجب عند استدعاء الطريقة أن تكون هنالك قيمة لمعامل القيمة الممرر للطريقة. يُعطي المترجم في المثال التالي:

```
public static bool TestEven(int n)
{
if (n % 2 == 0)
    return (true);
else
    return (false);
}
public static void Main( string[] args )
{

  int n;
 bool b;
 b=TestEven(n);
}
```

الخطأ التالي:

Use of unassigned local variable 'n'

(استخدام متغير بدون قيمة).

6. توافق ترتيب ونمط المعاملات:

يجب توافق ترتيب ونوع المعاملات عند الاستدعاء مع ترتيب ونوع المعاملات المعرفة. يُعطي المترجم في المثال التالي:

```
public static void Test(int n, bool b)
{

}
public static void Main( string[] args )
{
int n=1;
bool b=true;
Test(b , n);
}
```

الأخطاء الثلاثة التالية:

The best overloaded method match for 'Methods.Test(int, bool)' has some invalid arguments

Argument 1: cannot convert from 'bool' to 'int'Error

Argument 2: cannot convert from 'int' to 'bool'

2. أمثلة

الأهداف التعليمية:

أمثلة عن الطرق.

مثال 1: طباعة الأعداد الأولية المحصورة بين 1 و100

- ليكن المطلوب طباعة الأعداد الأولية المحصورة بين 1 و100 مثلاً.
- نقوم في المثال التالي بالتصريح عن الطريقة Prime والتي لها معامل دخل عدد طبيعي وتُعيد قيمة منطقية: true إذا كان العدد أولي و false إذا كان العدد غير أولي.
- لاحظ أننا نستخدم الخوارزمية التالية لاختبار أن عدد ما أولي أم لا: نفرض في البداية أن العدد أولي، ثم نقوم بقسمته على الأعداد من 2 إلى العدد نفسه مقسوماً على 2. في حال أن قبل العدد القسمة على أحد هذه الأعداد يكون إذاً العدد غير أولي.
- نقوم في الطريقة Main باستدعاء الطريقة السابقة Prime من أجل كل الأعداد بين 1 و 100.

```
using System;
public class Methods
{
    public static bool Prime(int N)
    {
        bool b;
        b = true;
        for (int i = 2; i <= N / 2; i++)
            if (N % i == 0)
            {
                b = false;
                break;
            }
        return b;
    }

   public static void Main( string[] args )
   {
       for (int i = 1; i < 100; i++)
           if (Prime(i))
             Console.WriteLine(i);
   } // end Main
}
```

يكون ناتج التنفيذ:

```
1
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Press any key to continue . . .
```

مثال 2: حساب القاسم المشترك الأعظم لعددين (1)

- ليكن المطلوب حساب القاسم المشترك الأعظم لعددين
- نقوم في المثال التالي بالتصريح عن الطريقة GCD والتي لها معاملي دخل من النمط أعداد صحيحة وتُعيد قيمة من النمط عدد صحيح
- لاحظ أننا نستخدم الخوارزمية التالية لحساب القاسم المشترك الأعظم لعددين: نكرر طرح العدد الصغير من العدد الكبير حتى يصبح العددان متساويين
- نقوم في الطريقة Main باستدعاء الطريقة السابقة بعد الطلب من المستخدم إدخال عددين ومن ثم نُظهر الناتج

```
using System;
public class Methods
{
    public static int GCD(int a, int b)
    {
        while (a != b)
            if (a > b)
                a = a - b;
            else
                b = b - a;
        return a;
    }

   public static void Main( string[] args )
   {
       int N, M, C;

       // prompt for and input two integer values
       Console.WriteLine("Enter two values,\n" +
          "  pressing 'Enter' after each one: ");
       N = Convert.ToInt32(Console.ReadLine());
       M = Convert.ToInt32(Console.ReadLine());

       C = GCD(N,M);

       // display result
       Console.WriteLine("GCD is: " + C);
   } // end Main
}
```

- يكون ناتج التنفيذ:

```
Enter two values,
  pressing 'Enter' after each one:
24
18
GCD is: 6
Press any key to continue . . .
```

مثال 3: حساب القاسم المشترك الأعظم لعددين (2)

- ليكن المطلوب حساب القاسم المشترك الأعظم لعددين
- نقوم في المثال التالي بالتصريح عن الطريقة GCD والتي لها معاملي دخل من النمط أعداد صحيحة وتُعيد قيمة من النمط عدد صحيح
- لاحظ أننا نستخدم الخوارزمية التالية لحساب القاسم المشترك الأعظم لعددين (خوارزمية إقليدس): نكرر قسمة العدد الكبير على العدد الصغير حتى نجد أن باقي القسمة صفر، فيكون العدد الصغير هو قيمة القاسم المشترك الأعظم
- نقوم في الطريقة Main باستدعاء الطريقة السابقة بعد الطلب من المستخدم إدخال عددين ومن ثم نُظهر الناتج

```
using System;
public class Methods
{
    public static int GCD(int a, int b)
    {
        int bigger, smaller, reminder;
        if (a > b)
        {
            bigger = a;
            smaller = b;
        }
        else
        {
            bigger = b;
            smaller = a;
        }
        reminder = bigger % smaller;
        while (reminder !=0)
        {
            bigger= smaller;
            smaller = reminder;
            reminder = bigger % smaller;
        }

        return smaller;
    }

   public static void Main( string[] args )
   {
       int N, M, C;

       // prompt for and input two integer values
       Console.WriteLine("Enter two values,\n" +
          "  pressing 'Enter' after each one: ");
       N = Convert.ToInt32(Console.ReadLine());
       M = Convert.ToInt32(Console.ReadLine());

       C = GCD(N,M);

       // display result
       Console.WriteLine("GCD is: " + C);
   } // end Main
}
```

- يكون ناتج التنفيذ:

```
Enter two values,
  pressing 'Enter' after each one:
24
18
GCD is: 6
Press any key to continue . . .
```

3. التمارين

تمرين:

قم بكتابة الطريقة LCM لحساب المضاعف المشترك البسيط لعددين صحيحين ثم استخدمه في تطبيق يطلب من المستخدم إدخال عددين صحيحين ثم يقوم بإظهار المضاعف المشترك البسيط لهما.

ملاحظة: يُمكنك استخدام الطريقة التالية لحساب المضاعف المشترك البسيط: كرر جمع القيمة الأولية للعدد الأصغر حتى يُصبح العددين متساويين.

مثال:

24	36
48	36
72	72

مواضيع متقدمة في الطرق

الكلمات المفتاحية:

مجال التصريح، التحميل الزائد للطرق، المعاملات الخيارية، المعاملات المسماة.

ملخص:

نتابع في هذا الفصل استعراض أساسيات التصريح عن الطرق واستخدامها.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- مجال التصريح
- التحميل الزائد للطرق
- المعاملات الخيارية
- المعاملات المسماة

المخطط:

يتضمن فصل "مواضيع متقدمة في الطرق" 3 وحدات عناوينها بالترتيب المحدد:

1. مجال التصريح.
2. التحميل الزائد للطرق.
3. المعاملات الخيارية والمعاملات المسماة.

1. مجال التصريح

الأهداف التعليمية:

مجال التصريح.

مجال التصريح Scope of Declarations:

- يكون لكل متغير مجال (scope) أي مجموعة التعليمات التي يُمكن لها أن تتعامل مع هذا المتغير
- تُبين القواعد التالية أساسيات مجال التصريح:

1. يكون مجال معاملات الطريقة جسم الطريقة نفسها فقط.
2. يكون مجال متغير محلي (local variable) مُصرح عنه ضمن الطريقة كتلة التعليمات (block) التي تحوي التصريح.
3. يكون مجال المتغيرات المُصرح عنها ضمن قسم التهيئة الابتدائية في التعليمة for جسم التعليمة for وأي تعابير مستخدمة في ترويستها.
4. يكون مجال الطريقة المُصرح عنها ضمن صف ما كامل جسم الصف هذا الصف.
5. يُمكن للطريقة الساكنة (static) أن تتعامل فقط مع حقول الصف الساكنة.
6. يُمكن لكل كتلة أن تُصرّح عن متغيراتها.
7. يتم العودة للمتغير المُصرح عنه في الكتلة الأقرب.

نُصرّح في المثال التالي عن عدة متغيرات لها نفس الاسم (x):

1. نُصرّح أولاً في جسم الصف عن المتغير الساكن(x). يُمكن لجميع طرق هذا الصف الوصول لهذا المتغير والتعامل معه.
2. نُصرّح في الطريقة الساكنة Main عن متغير له أيضاً الاسم (x). سيقوم هذا المتغير بحجب المتغير السابق الساكن في جسم الطريقة Main، بمعنى أن أي استخدام للمتغير (x) في جسم الطريقة Main سيعود إلى المتغير (x) المُصرّح عنه ضمن هذه الطريقة.
3. نُصرّح في الطريقة `UseLocalVariable()` عن المتغير المحلي (x). سيعود أي استخدام للمتغير (x) ضمن جسم هذه الطريقة إلى هذا المتغير. يُعاد تهيئة هذا المتغير المحلي عند كل استدعاء للطريقة. بمعنى أنه في كل مرة نستدعي الطريقة `UseLocalVariable()` ستكون قيمة المتغير (x) الابتدائية 25.
4. نستخدم في الطريقة `UseStaticVariable()` المتغير (x). وبما أننا لم نصرح ضمن هذه الطريقة عن متغير باسم (x) سيعود استخدام (x) ضمن هذه الطريقة إلى المتغير (x) المُصرّح عنه ضمن جسم الصف.

```
// Scope.cs
// Scope class demonstrates static and local variable scopes.
using System;

public class Scope
{
   // static variable that is accessible to all methods of this class
   private static int x = 1;

   // Main creates and initializes local variable x
   // and calls methods UseLocalVariable and UseStaticVariable
   public static void Main( string[] args )
   {
      int x = 5; // method's local variable x hides static variable x

      Console.WriteLine( "local x in method Main is {0}", x );

      // UseLocalVariable has its own local x
      UseLocalVariable();

      // UseStaticVariable uses class Scope's static variable x
      UseStaticVariable();

      // UseLocalVariable reinitializes its own local x
      UseLocalVariable();

      // class Scope's static variable x retains its value
      UseStaticVariable();

      Console.WriteLine( "\nlocal x in method Main is {0}", x );
   } // end Main

   // create and initialize local variable x during each call
   public static void UseLocalVariable()
   {
      int x = 25; // initialized each time UseLocalVariable is called

      Console.WriteLine(
         "\nlocal x on entering method UseLocalVariable is {0}", x );
      ++x; // modifies this method's local variable x
      Console.WriteLine(
         "local x before exiting method UseLocalVariable is {0}", x );
   } // end method UseLocalVariable

   // modify class Scope's static variable x during each call
   public static void UseStaticVariable()
   {
      Console.WriteLine( "\nstatic variable x on entering {0} is {1}",
         "method UseStaticVariable", x );
      x *= 10; // modifies class Scope's static variable x
      Console.WriteLine( "static variable x before exiting {0} is {1}",
         "method UseStaticVariable", x );
   } // end method UseStaticVariable
} // end class Scope
```

- يكون ناتج التنفيذ:

```
local x in method Main is 5

local x on entering method UseLocalVariable is 25
local x before exiting method UseLocalVariable is 26

static variable x on entering method UseStaticVariable is 1
static variable x before exiting method UseStaticVariable is 10

local x on entering method UseLocalVariable is 25
local x before exiting method UseLocalVariable is 26

static variable x on entering method UseStaticVariable is 10
static variable x before exiting method UseStaticVariable is 100

local x in method Main is 5
Press any key to continue . . .
```

2. التحميل الزائد للطرق

الأهداف التعليمية:

التحميل الزائد للطرق.

التحميل الزائد للطرق Method Overloading:

- يُمكن في الصف الواحد التصريح عن عدة طرق بنفس الاسم طالما أن لكل منها مجموعة مختلفة من المعاملات (عدد المعاملات وأنماط المعاملات). تُدعى هذه الميزة بالتحميل الزائد للطرق method overloading
- عندما يتم استدعاء طريقة مُحمّله، يقوم المترجم بانتقاء الطريقة الموافقة عن طريق فحص عدد المعاملات وأنماطها وترتيبها
- يتم عادةً استخدام التحميل الزائد عندما نحتاج لإنجاز نفس المهمة إنما على أنماط مختلفة. فمثلاً، يكون للطريقة Max في الصف Math 11 نسخة (تختلف بأنماط المعاملات)

مثال: نقوم في المثال التالي بالتصريح عن طريقتين لهما نفس الاسم Square. تقبل النسخة الأولى من الطريقة عدد صحيح وتُعيد مربع هذا العدد كعدد صحيح. بينما تقبل النسخة الثانية عدد عشري وتُعيد مربع هذا العدد (عدد عشري)

```
// MethodOverload.cs
// Overloaded method declarations.
using System;

public class MethodOverload
{
   // test overloaded square methods
   public static void Main( string[] args )
   {
      Console.WriteLine( "Square of integer 7 is {0}", Square( 7 ) );
      Console.WriteLine( "Square of double 7.5 is {0}", Square( 7.5 ) );
   } // end Main

   // square method with int argument
   public static int Square( int intValue )
   {
      Console.WriteLine( "Called square with int argument: {0}", intValue );
      return intValue * intValue;
   } // end method Square with int argument

   // square method with double argument
   public static double Square( double doubleValue )
   {
      Console.WriteLine( "Called square with double argument: {0}",
         doubleValue );
      return doubleValue * doubleValue;
   } // end method Square with double argument
} // end class MethodOverload
```

- يكون ناتج التنفيذ:

```
Called square with int argument: 7
Square of integer 7 is 49
Called square with double argument: 7.5
Square of double 7.5 is 56.25
Press any key to continue . . .
```

القيمة المُعادة والتحميل الزائد:

لا يُمكن التفريق بين الطرق عن طريق نمط القيمة المعادة فقط. بمعنى أنه لا يُسمح بتعريف طريقتين لهما نفس الاسم ونفس المعاملات وإنما تختلفان بالقيمة المُعادة

يُعيد المترجم في المثال التالي:

```
// MethodOverload.cs
// Overloaded methods with identical signatures
// cause compilation errors, even if return types are different.
public class MethodOverloadError
{
   // declaration of method Square with int argument
   public static int Square( int x )
   {
      return x * x;
   } // end method Square

   // second declaration of method Square with int argument
   // causes compilation error even though return types are different
   public static double Square( int y )
   {
      return y * y;
   } // end method Square
} // end class MethodOverloadError
```

رسالة الخطأ التالية:

```
Type 'MethodOverloadError' already defines a member called 'Square' with the same parameter types
```

3. المعاملات الخيارية والمعاملات المسماة

الأهداف التعليمية:

المعاملات الخيارية والمعاملات المسماة.

المعاملات الخيارية Optional Parameters:

- يُمكن أن يكون للطريقة معاملات خيارية، بمعنى أنه يُمكن استدعاء الطريقة وبدون تمرير قيم لهذه المعاملات
- تُعطى المعاملات الخيارية قيم افتراضية لإسنادها لهذه المتغيرات في حال عدم تمرير قيم لها عند الاستدعاء
- يُمكن أن يكون للطريقة عدة معاملات خيارية
- يجب وضع المعاملات الخيارية في نهاية قائمة المعاملات أي بعد المعاملات الوتاجب تمرير قيم لها عند الاستدعاء

نقوم في المثال التالي بالتصريح عن معامل خياري للطريقة Power. في حال عدم تمرير قيمة لهذه الطريقة عند الاستدعاء، سيُعطى هذا المعامل القيمة 2.

```
// Optional argument demonstration with method Power.
using System;

class CalculatePowers
{
   // call Power with and without optional arguments
   public static void Main( string[] args )
   {
      Console.WriteLine( "Power(10) = {0}", Power( 10 ) );
      Console.WriteLine( "Power(2, 10) = {0}", Power( 2, 10 ) );
   } // end Main

   // use iteration to calculate power
   public static int Power( int baseValue, int exponentValue = 2 )
   {
      int result = 1; // initialize total

      for ( int i = 1; i <= exponentValue; i++ )
         result *= baseValue;

      return result;
   } // end method Power
} // end class CalculatePowers
```

- يكون ناتج التنفيذ:

```
Power(10) = 100
Power(2, 10) = 1024
Press any key to continue . . .
```

المعاملات المُسماة Named Parameters:

- عند التصريح عن عدة معاملات خيارية وتمرير بعض القيم عند الاستدعاء، فإن القيم ستُمرر إلى المعاملات الخيارية من اليسار لليمين بالترتيب
- فمثلاً، نقوم في الطريقة التالية بالتصريح عن 3 معاملات خيارية:

```
public static int getSeconds(int hour = 0, int minute = 0, int second = 0)
    {
        return hour * 3600 + minute * 60 + second;
    }
```

- يُمكن استدعاء الطريقة السابقة بالأشكال المختلفة التالية:

```
public static void Main( string[] args )
   {
       int s;

       s = getSeconds();
       Console.WriteLine("Seconds = {0} ", s);
       s = getSeconds(12);
       Console.WriteLine("Seconds = {0} ", s);
       s = getSeconds(12, 30);
       Console.WriteLine("Seconds = {0} ", s);
       s = getSeconds(12, 30, 22);
       Console.WriteLine("Seconds = {0} ", s);

       s = getSeconds(hour: 1, second: 30);
       Console.WriteLine("Seconds = {0} ", s);

   } // end Main
```

- يكون الناتج:

```
Seconds = 0
Seconds = 43200
Seconds = 45000
Seconds = 45022
Seconds = 3630
Press any key to continue . . .
```

- في حال نريد تمرير قيمة للساعة والثواني بدون تمرير قيمة للدقائق سيكون من الخطأ كتابة:

```
s=getSeconds (1, , 30);
```

- حيث يُعطي المترجم رسالة الخطأ التالية:

Argument missing

- يُمكن استخدام المعاملات المسماة لتحقيق ذلك بكتابة:

```
s=getSeconds (hour:1,second:30);
Console.WriteLine("Seconds = {0} ", s);
```

- يُعطي التنفيذ:

```
Seconds = 3630
```

4. التمارين

تمرين 1:

قم بكتابة عدة نسخ من الطريقة Average بحيث تقبل الطريقة ثلاثة أعداد من النمط int أو long أو double وتُعيد الوسطي الحسابي لها.

تمرين 2:

عدّل الطرق السابقة لحساب الوسطي الحسابي. بحيث يُمكن تمرير معامل واحد أو معاملين أو ثلاثة معاملات للطريقة.

تمرين 3:

نقول عن عدد أنه كامل إذا كان مجموع عوامله يساوي العدد نفسه. مثلاً:

$$6 = 1 + 2 + 3$$

قم بكتابة الطريقة Perfect لاختبار فيما إذا كان عدد كامل أم لا.

قم بعدها باستدعاء هذه الطريقة على جميع الأعداد من 2 إلى 1000 لطباعة الأعداد الكاملة ضمن هذا المجال.

تمرين 4:

قم بكتابة طريقة تقبل عدد صحيح كمعامل دخل وتُعيد معكوس هذا العدد. مثلاً، إذا كان العدد 7631 فإن الطريقة تُعيد 1367.

العودية Recursion

الكلمات المفتاحية:

العودية Recursion.

ملخص:

نتعرض في هذا الفصل لمفهوم واستخدام العودية.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- استخدام العودية Recursion
- أمثلة شهيرة على استخدام العودية

المخطط:

يتضمن فصل "العودية Recursion" وحدتين عناوينها بالترتيب المحدد:

1. العودية Recursion.
2. مثال تعليمي: أبراج هانوي.

1. العودية Recursion

الأهداف التعليمية:

العودية Recursion.

العودية:

- يُمكن لطريقة أن تستدعي نفسها. ندعو هذه لتقية بالعودية.
- تقوم الطريقة التالية `f()` مثلاً بطباعة جملة ثم استدعاء نفسها:

```
public static void f()
{
  Console.WriteLine("Welcome to infinite recursion");
  f();
}
```

- سيؤدي استدعاء الطريقة السابقة إلى طباعة الجملة عدد كبير من المرات:

```
public static void Main( string[] args )
   {
       f();
   } // end Main
```

```
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion
Welcome to infinite recursion

Process is terminated due to StackOverflowException.
```

- لاحظ في النهاية ظهور الخطأ StackOverflowException والذي يُعلم بملء الذاكرة المخصصة (المكدّس Stack).

شرط توقف العودية:

- نضع عادةً في الطرق العودية شرط لتوقف العودية.
- تختبر الإجرائية العودية التالية قيمة المعامل n فيما إذا كان أكبر من الصفر لإعادة استدعاء الإجرائية:

```
using System;
public class Recursion
{
    public static void CountDown(int n)
    {
        Console.WriteLine(n);
        if (n > 1)
            CountDown(n - 1);
    }
   public static void Main( string[] args )
   {
       CountDown(10);
   } // end Main
} // end class
```

- يكون ناتج التنفيذ:

```
10
9
8
7
6
5
4
3
2
1
Press any key to continue . . .
```

شرط توقف العودية:

نُعطي فيما يلي مثال آخر:

```
using System;
public class Recursion
{
   public static void CountUp(int n)
    {
        if (n > 1)
            CountUp(n - 1);
        Console.WriteLine(n);
    }
   public static void Main( string[] args )
   {
       CountUp(10);
   } // end Main
   // recursive declaration of method Factorial
} // end class
```

- يكون ناتج التنفيذ:

```
1
2
3
4
5
6
7
8
9
10
Press any key to continue . . .
```

مثال تعليمي: حساب العاملي لعدد صحيح:

يُعرّف العاملي لعدد صحيح n بالعلاقة العودية: $n! = n \cdot (n-1)!$

فمثلاً يكون العاملي للعدد 5 مساوياً إلى : $5! = 5 \cdot (4!)$

- نقوم في التطبيق التالي بكتابة الطريقة العودية `Factorial` لحساب قيمة العاملي لعدد n:

```
// FactorialTest.cs
// Recursive Factorial method.
using System;
public class FactorialTest
{
    public static long Factorial(long number)
    {
        // base case
        if (number <= 1)
            return 1;
        // recursion step
        else
            return number * Factorial(number - 1);
    } // end method Factorial
   public static void Main( string[] args )
   {
      // calculate the factorials of 0 through 10
      for ( long counter = 0; counter <= 10; ++counter )
         Console.WriteLine( "{0}! = {1}",
            counter, Factorial( counter ) );
   } // end Main

   // recursive declaration of method Factorial

} // end class FactorialTest
```

- نستدعي في الطريقة Main الطريقة Factorial من أجل الأعداد من 0 إلى 10.
- يكون ناتج التنفيذ:

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
Press any key to continue . . .
```

2. مثال تعليمي: أبراج هانوي

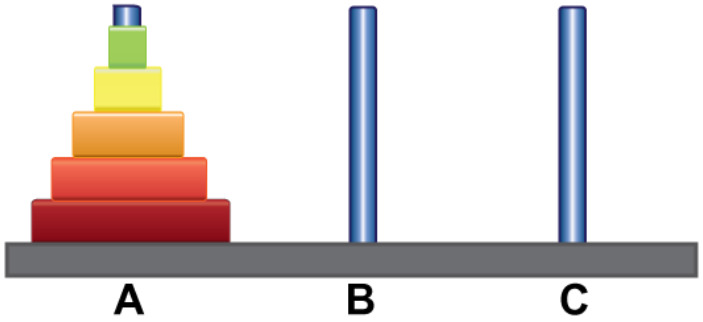
الأهداف التعليمية:

مثال تعليمي.

مثال: أبراج هانوي:

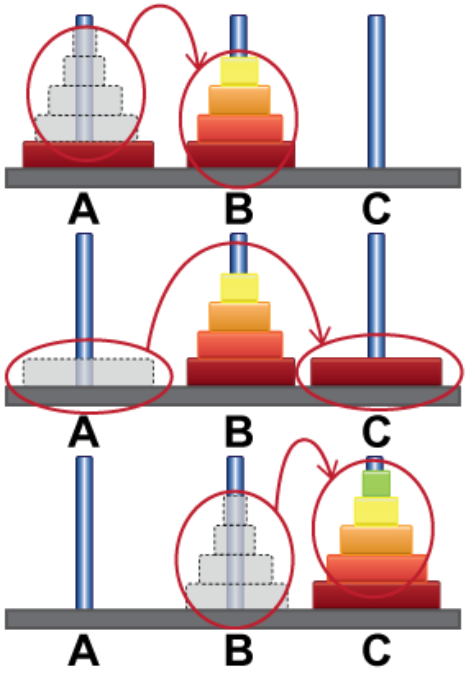
ليكن المطلوب في هذه اللعبة نقل n قرص من البرج الأول(المصدر) إلى البرج الثالث (الوجهة). يجب احترام القواعد التالية للعبة:

1. لا يُمكن نقل إلا قرص واحد بنفس لوقت.
2. لا يجوز وضع قرص فوق قرص أكبر منه.
3. يُمكن وضع أقراص على البرج الثاني (الوسيط).



- يعتمد حل هذه اللعبة على المبدأ العودي التالي:

1. قم أولاً بحل لمسألة التالية: نقل n − 1 قرص من البرج المصدر إلى البرج الوسيط.
2. قم بنقل القرص المتبقي في البرج المصدر إلى البرج الوجهة.
3. قم بحل المسألة التالية: نقل n − 1 قرص من البرج الوسيط إلى البرج الوجهة.



يُمكن ترجمة الخطوات السابقة في الطريقة العودية Hanoi التالية:

```
using System;
public class Recursion
{
    public static void Hanoi(int n, char first, char last, char temp)
    {
        if (n > 1)
        {
            Hanoi(n - 1, first, temp, last);
            Hanoi(1, first, last, temp);
            Hanoi(n - 1, temp, last, first);
        }
        else
            Console.WriteLine(" Move a disk from {0}  to  {1} ", first, last);
    }

   public static void Main( string[] args )
   {
       Hanoi(3, 'A', 'C', 'B') ;
   } // end Main

} // end class
```

- يكون ناتج التنفيذ من أجل ثلاثة أقراص مثلاً:

```
 Move a disk from A  to  C
 Move a disk from A  to  B
 Move a disk from C  to  B
 Move a disk from A  to  C
 Move a disk from B  to  A
 Move a disk from B  to  C
 Move a disk from A  to  C
Press any key to continue . . .
```

3. التمارين

تمرين 1:

قم بكتابة طريقة عودية لحساب القوة n لعدد x. مع ملاحظة:

$$Power(x, 0) = 1$$

$$Power(x, n) = x * Power(x, n-1)$$

تمرين 2:

قم بكتابة طريقة عودية لحساب القاسم المشترك الأعظم لعددين مستخدماً الخوارزمية التالية: كرر طرح العدد الصغير من الكبير حتى يصبح العددان متساويين.

تمرين 3:

قم بكتابة طريقة عودية لحساب القاسم المشترك الأعظم لعددين مستخدماً الخوارزمية التالية: كرر قسمة العدد الكبير على العدد الصغير حتى يصبح باقي القسمة صفر.

طرق تمرير المعاملات

الكلمات المفتاحية:

التمرير بالقيمة، التمرير بالمرجع، معاملات الخرج.

ملخص:

نعرض في هذا الفصل مختلف طرق تمرير المعاملات إلى الطرق.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- معاملات القيمة
- معاملات المرجع
- معاملات الخرج

المخطط:

يتضمن فصل "طرق تمرير المعاملات" وحدتين عناوينها بالترتيب المحدد:

1. طرق تمرير المعاملات.
2. أمثلة.

1. طرق تمرير المعاملات

الأهداف التعليمية:

طرق تمرير المعاملات.

طرق تمرير المعاملات:

يوجد ثلاثة طرق لتمرير معاملات الطرق:

1. **التمرير بالقيمة value:**

1. يتم في هذه الحالة تمرير نسخة من قيمة المتغير الممرر كمعامل إلى الطريقة.
2. لن يتأثر هذا المتغير بأي تعديلات تقوم بها الطريقة على المعامل في جسم الطريقة.

2. **التمرير بالمرجع reference:**

1. يتم في هذه الحالة تمرير مرجع المتغير (عنوان) الممرر كمعامل إلى الطريقة.
2. يتأثر هذا المتغير بأي تعديلات تقوم بها الطريقة على المعامل في جسم الطريقة.
3. يجب أن يكون لمتغير الممرر مهيئاً (له قيمة).

3. **معاملات الخرج out:**

1. يتم في هذه الحالة تمرير مرجع المتغير (عنوان) الممرر كمعامل إلى الطريقة.
2. يتأثر هذا المتغير بأي تعديلات تقوم بها الطريقة على المعامل في جسم الطريقة.
3. يُمكن أن يكون المتغير لممرر غير مهيئ (ليس له قيمة).

يُبين المثال التالي مختلف الحالات السابقة:

```
// ReferenceAndOutputParameters.cs
// Reference, output and value parameters.
using System;
class ReferenceAndOutputParameters
{
    // uses reference parameter x to modify caller's variable
    static void SquareRef(ref int x)
    {
        x = x * x; // squares value of caller's variable
    } // end method SquareRef

    // uses output parameter x to assign a value
    // to an uninitialized variable
    static void SquareOut(out int x)
    {
        x = 6; // assigns a value to caller's variable
        x = x * x; // squares value of caller's variable
    } // end method SquareOut

    // parameter x receives a copy of the value passed as an argument,
    // so this method cannot modify the caller's variable
    static void Square(int x)
    {
        x = x * x;
    } // end method Square

   // call methods with reference, output and value parameters
   public static void Main( string[] args )
   {
      int y = 5; // initialize y to 5
      int z; // declares z, but does not initialize it

      // display original values of y and z
      Console.WriteLine( "Original value of y: {0}", y );
      Console.WriteLine( "Original value of z: uninitialized\n" );

      // pass y and z by reference
      SquareRef( ref y ); // must use keyword ref
      SquareOut( out z ); // must use keyword out

      // display values of y and z after they are modified by
      // methods SquareRef and SquareOut, respectively
      Console.WriteLine( "Value of y after SquareRef: {0}", y );
      Console.WriteLine( "Value of z after SquareOut: {0}\n", z );

      // pass y and z by value
      Square( y );
      Square( z );

      // display values of y and z after they are passed to method Square
      // to demonstrate arguments passed by value are not modified
      Console.WriteLine( "Value of y after Square: {0}", y );
      Console.WriteLine( "Value of z after Square: {0}", z );
   } // end Main


} // end class ReferenceAndOutputParameters
```

- يكون ناتج التنفيذ:

```
Original value of y: 5
Original value of z: uninitialized

Value of y after SquareRef: 25
Value of z after SquareOut: 36

Value of y after Square: 25
Value of z after Square: 36
Press any key to continue . . .
```

2. أمثلة

الأهداف التعليمية:

أمثلة.

كتابة طريقة للتبديل بين قيمتي متغيرين :

مثال: كتابة طريقة للتبديل بين قيمتي متغيرين.

- نقوم في المثال التالي بكتابة نسخة أولى من الطريقة swap للتبديل بين قيمتي متغيرين. تستخدم النسخة الأولى طريقة التمرير بالقيمة (الطريقة الافتراضية)
- نلاحظ أنه عند استدعاء هذه النسخة لن تتأثر قيم المتغيرات الممررة بالقيمة لها
- نقوم بكتابة نسخة ثانية من الطريقة swap تستخدم طريقة التمرير بالمرجع (ref)
- نلاحظ أنه عند استدعاء هذه النسخة الثانية مع تمرير مرجع المتغيرات كمعاملات لها بتغيير قيم المتغيرات الممررة

```
using System;
class ReferenceAndOutputParameters
{
   static void swap ( int n,   int m)
{
        int k;
        k=n;
        n=m;
        m=k;
}
   static void swap(ref int n, ref int m)
{
        int k;
        k=n;
        n=m;
        m=k;
}
   public static void Main( string[] args )
   {
        int x = 1, y = 10;
        Console.WriteLine("Before swap x= {0}, y= {1}", x, y);
        swap(x, y);
        Console.WriteLine("After swap(Passing values) x= {0}, y= {1}", x, y);
        swap( ref x, ref y);
        Console.WriteLine("After swap(Passing references) x= {0}, y= {1}", x, y);

   } // end Main
}
```

- يكون ناتج التنفيذ:

```
Before swap x= 1, y= 10
After swap( Passing values) x= 1, y= 10
After swap(Passing references) x= 10, y= 1
Press any key to continue . . .
```

كتابة طريقة لإيجاد القيمة الأكبر والأصغر والوسطي الحسابي لعددين:

مثال: كتابة طريقة لإيجاد القيمة الأكبر والأصغر والوسطي الحسابي لعددين.

- نقوم في المثال التالي بكتابة طريقة واحدة لإيجاد القيمة الأكبر والأصغر والوسطي الحسابي لعددين
- لاحظ أننا نستخدم ثلاثة معاملات خرج (out)
- لاحظ أنه عند الاستدعاء نمرر للطريقة ثلاثة متغيرات غير مهيأة (بدون قيم ابتدائية)

```
using System;
class ReferenceAndOutputParameters
{
   static void MaxMinAvg(double x, double y, out double max, out double min, out
double avg)
   {
       if (x > y)
       {
           max = x;
           min = y;
       }
       else
       {
           max = y;
           min = x;
       }
       avg = (x + y) / 2;
   }

   public static void Main( string[] args )
   {
       double a = 10;
       double b = 20;
       double mx, mn, av;
       MaxMinAvg(a, b, out mx, out mn, out av);
       Console.WriteLine("Max= {0}, Min= {1}, Avg={2} ", mx, mn, av);

   } // end Main
}
```

- يكون ناتج التنفيذ:

```
Max= 20, Min= 10, Avg=15
Press any key to continue . . .
```

3. التمارين

تمرين:

قم بكتابة طريقة نمرر لها 3 أعداد فتُعيد القيمة الأكبر والقيمة الأصغر من هذه الأعداد.

المصفوفات (1)

الكلمات المفتاحية:

التصريح عن المصفوفات، إنشاء المصفوفات، foreach.

ملخص:

ندرس في هذا الفصل آليات التعامل مع المصفوفات الأحادية.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- التصريح عن المصفوفات
- إنشاء المصفوفات
- التعليمة التكرارية foreach
- تمرير المصفوفات كمعاملات دخل

المخطط:

يتضمن فصل "المصفوفات (1)" 3 وحدات عناوينها بالترتيب المحدد:

1. التصريح وإنشاء المصفوفات.
2. استخدام التعليمة foreach.
3. تمرير المصفوفات كمعاملات للطرق.

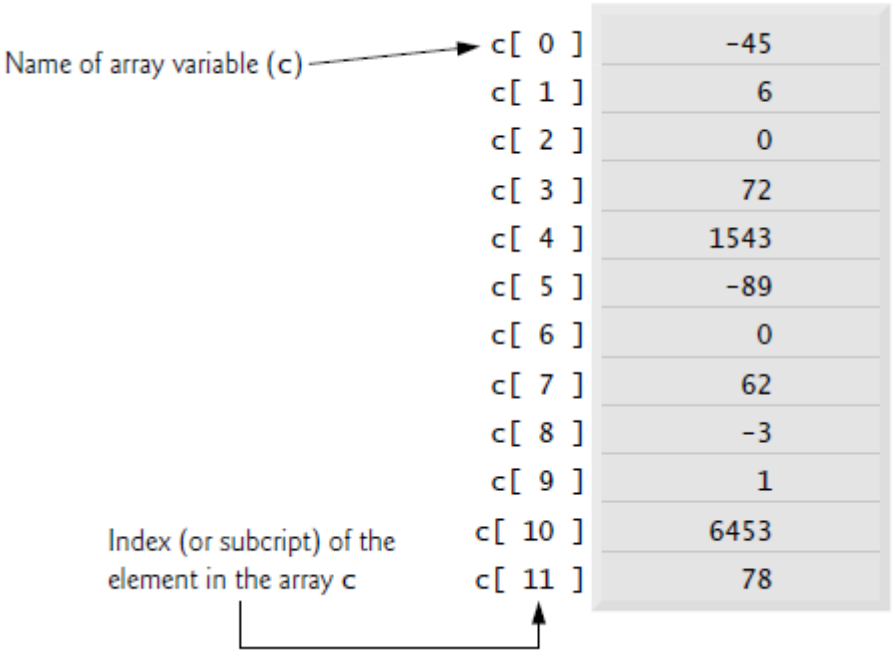
1. التصريح وإنشاء المصفوفات

الأهداف التعليمية:

التصريح وإنشاء المصفوفات.

المصفوفات:

- تُعرّف لمصفوفة بأنها مجموعة من العناصر من نفس النمط والتي لها نفس الاسم.
- يكون لكل عنصر فهرس index.
- تبدأ فهرسة العنصر الأول من الصفر.
- تُعطي الخاصية Length عدد عناصر المصفوفة.
- يُبين الشكل التالي مصفوفة من الأعداد لطبيعية تحوي 12 عنصر:



التصريح وإنشاء المصفوفات:

- يتم التصريح عن مصفوفة من نمط معين كما يلي:

```
TypeName [] arrayName;
```

- لا تقوم التعليمة السابقة بإنشاء العناصر وحجز مكانها في الذاكرة. للقيام بذلك يجب استخدام الكلمة المفتاحية new وتحديد عدد العناصر:

```
arrayName = new TypeName [arrayLength]
```

- يُمكن أيضاً التصريح عن المصفوفة وإنشاء عناصرها كما في المثال التالي:

```
int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
```

- تُبين الأمثلة التالية مختلف إمكانات التصريح وإنشاء المصفوفات.

مثال 1:

نقوم في هذا المثال بالتصريح عن المصفوفة array من النمط int. ثم استخدام الكلمة المفتاحية new لإنشاء خمسة عناصر. سيتم إعطاء جميع عناصر المصفوفة القيمة الابتدائية 0

```
// InitArray.cs
// Creating an array.
using System;
public class InitArray
{
   public static void Main( string[] args )
   {
      int[] array; // declare array named array

      // create the space for array and initialize to default zeros
      array = new int[ 5 ]; // 5 int elements

      Console.WriteLine( "{0}{1,8}", "Index", "Value" ); // headings

      // output each array element's value
      for ( int counter = 0; counter < array.Length; ++counter )
         Console.WriteLine( "{0,5}{1,8}", counter, array[ counter ] );
   } // end Main
} // end class InitArray
```

- يكون ناتج التنفيذ:

```
Index   Value
  0       0
  1       0
  2       0
  3       0
  4       0
Press any key to continue . . .
```

مثال2:

نقوم في هذا المثال بالتصريح عن المصفوفة array من النمط int. وإعطاء قيم عناصرها مباشرة:

```
// InitArray.cs
// Initializing the elements of an array with an array initializer.
using System;

public class InitArray
{
   public static void Main( string[] args )
   {
      // initializer list specifies the value for each element
      int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

      Console.WriteLine( "{0}{1,8}", "Index", "Value" ); // headings

      // output each array element's value
      for ( int counter = 0; counter < array.Length; ++counter )
         Console.WriteLine( "{0,5}{1,8}", counter, array[ counter ] );
   } // end Main
} // end class InitArray
```

- يكون ناتج التنفيذ:

```
Index   Value
  0      32
  1      27
  2      64
  3      18
  4      95
  5      14
  6      90
  7      70
  8      60
  9      37
Press any key to continue . . .
```

مثال3:

نقوم في هذا المثال بالتصريح عن الثابت ARRAY_LENGTH ومن ثم التصريح عن المصفوفة array من النمط int وإنشاء ARRAY_LENGTH عنصر.

نقوم بعدها بالدوران على عناصر المصفوفة لإعطاء كل عنصر قيمة ابتدائية.

```
// InitArray.cs
// Calculating values to be placed into the elements of an array.
using System;

public class InitArray
{
   public static void Main( string[] args )
   {
      const int ARRAY_LENGTH = 10; // create a named constant
      int[] array = new int[ ARRAY_LENGTH ]; // create array

      // calculate value for each array element
      for ( int counter = 0; counter < array.Length; ++counter )
         array[ counter ] = 2 + 2 * counter;

      Console.WriteLine( "{0}{1,8}", "Index", "Value" ); // headings

      // output each array element's value
      for ( int counter = 0; counter < array.Length; ++counter )
         Console.WriteLine( "{0,5}{1,8}", counter, array[ counter ] );
   } // end Main
} // end class InitArray
```

- يكون ناتج التنفيذ:

```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
Press any key to continue . . .
```

2. استخدام التعليمة foreach

الأهداف التعليمية:

تعليمة التكرار foreach.

التعليمة التكرارية foreach:

نقوم عادةً بالدوران على عناصر مصفوفة باستخدام التعليمة for مع عداد للفهرس يبدأ من الصفر إلى طول المصفوفة ناقص واحد.

مثال: نقوم في المثال التالي بالدوران على جميع عناصر المصفوفة لإيجاد مجموع هذه العناصر.

```
// SumArray.cs
// Computing the sum of the elements of an array.
using System;

public class SumArray
{
   public static void Main( string[] args )
   {
      int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for ( int counter = 0; counter < array.Length; ++counter )
         total += array[ counter ];

      Console.WriteLine( "Total of array elements: {0}", total );
   } // end Main
} // end class SumArray
```

- يكون الناتج:

```
Total of array elements: 849
Press any key to continue . . .
```

تسمح التعليمة التكرارية foreach بالدوران على عناصر المصفوفة دون الحاجة لتحديد الفهرس.

مثال: يُبين المثال التالي استخدام التعليمة foreach لجمع عناصر المصفوفة.

```
// ForEachTest.cs
// Using the foreach statement to total integers in an array.
using System;

public class ForEachTest
{
   public static void Main( string[] args )
   {
      int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      foreach ( int number in array )
         total += number;

      Console.WriteLine( "Total of array elements: {0}", total );
   } // end Main
} // end class ForEachTest
```

- يكون الناتج:

```
Total of array elements: 849
Press any key to continue . . .
```

3. تمرير المصفوفات كمعاملات للطرق

الأهداف التعليمية:

تمرير المصفوفات كمعاملات للطرق.

تمرير المصفوفات كمعاملات للطرق:

- عند تمرير مصفوفة كمعامل دخل لطريقة فإن التمرير يكون وفق المرجع بمعنى أن أي تعديل على عناصر المصفوفة ضمن الطريقة سيؤدي إلى تأثر عناصر المصفوفة الممررة.
- عند تمرير عنصر من المصفوفة كمعامل دخل لطريقة، يُعامل هذا العنصر مثل أي متغير آخر أي يُمكن أن يمرر بالقيمة أو بالمرجع أو أن يكون متغير خرج.
- نقوم في المثال التالي بالتصريح عن الطريقة `ModifyArray` والتي لها معامل دخل مصفوفة. نقوم في داخل الطريقة بالدوران على عناصر المصفوفة لضربهم بـ 2.
- لاحظ أنه عند استدعاء الطريقة السابقة وتمرير مصفوفة لها. ستتأثر عناصر المصفوفة الممرة بعد الاستدعاء (تُضرب بـ 2).
- نقوم في المثال التالي بالتصريح عن الطريقة `ModifyElement` والتي لها معامل دخل صحيح. نقوم في داخل الطريقة بضرب المعامل الممرر بـ 2.
- لاحظ أنه عند استدعاء الطريقة السابقة وتمرير عنصر من المصفوفة كمعامل لها. لن يتأثر عنصر المصفوفة (معامل قيمة).

```
// PassArray.cs
// Passing arrays and individual array elements to methods.
using System;

public class PassArray
{
   // Main creates array and calls ModifyArray and ModifyElement
   public static void Main( string[] args )
   {
      int[] array = { 1, 2, 3, 4, 5 };

      Console.WriteLine(
         "Effects of passing reference to entire array:\n" +
         "The values of the original array are:" );

      // output original array elements
      foreach ( int value in array )
         Console.Write( "   {0}", value );

      ModifyArray( array ); // pass array reference
      Console.WriteLine( "\n\nThe values of the modified array are:" );

      // output modified array elements
      foreach ( int value in array )
         Console.Write( "   {0}", value );

      Console.WriteLine(
         "\n\nEffects of passing array element value:\n" +
         "array[3] before ModifyElement: {0}", array[ 3 ] );

      ModifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
      Console.WriteLine(
         "array[3] after ModifyElement: {0}", array[ 3 ] );
   } // end Main

   // multiply each element of an array by 2
   public static void ModifyArray( int[] array2 )
   {
      for ( int counter = 0; counter < array2.Length; ++counter )
         array2[ counter ] *= 2;
   } // end method ModifyArray

   // multiply argument by 2
   public static void ModifyElement( int element )
   {
      element *= 2;
      Console.WriteLine(
         "Value of element in ModifyElement: {0}", element );
   } // end method ModifyElement
} // end class PassArray
```

- يكون ناتج التنفيذ:

```
Effects of passing reference to entire array:
The values of the original array are:
  1   2   3   4   5

The values of the modified array are:
  2   4   6   8   10

Effects of passing array element value:
array[3] before ModifyElement: 8
Value of element in ModifyElement: 16
array[3] after ModifyElement: 8
Press any key to continue . . .
```

4. التمارين

تمرين1:

قم بكتابة طريقة نُمرر لها مصفوفة أحادية من الأعداد فتقوم بحساب أكبر قيمة، أصغر قيمة، وسطي القيم (استخدم معاملات خرج out). قم باختبار هذه الطريقة في تطبيق.

تمرين2:

قم بكتابة طريقة نُمرر لها مصفوفة أحادية من الأعداد فتقوم بإيجاد العنصر ذو التواتر الأكبر في المصفوفة وعدد مرات تواتره (استخدم معاملات خرج out). قم باختبار هذه الطريقة في تطبيق.

تمرين3:

استخدم مصفوفة أحادية لحل مشكلة التكرار التالية:

اكتب تطبيق يقوم بقراءة 10 قيم مدخلة من قبل المستخدم ومن ثم طباعة القيم غير المكررة فقط.

المصفوفات (2)

الكلمات المفتاحية:

المصفوفات المستطيلة Rectangular Arrays، المصفوفات المسننة Jagged Arrays.

ملخص:

نتعلم في هذا الفصل التعامل مع المصفوفات الثنائية.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- المصفوفات المستطيلة
- المصفوفات المسننة
- قائمة المعاملات متغيرة الطول .

المخطط:

يتضمن فصل "المصفوفات (2)" 3 وحدات عناوينها بالترتيب المحدد:

1. المصفوفات الثنائية.
2. مثال تعليمي.
3. قائمة المعاملات متغيرة الطول

1. المصفوفات الثنائية

الأهداف التعليمية:

التعامل مع المصفوفات الثنائية.

المصفوفات الثنائية:

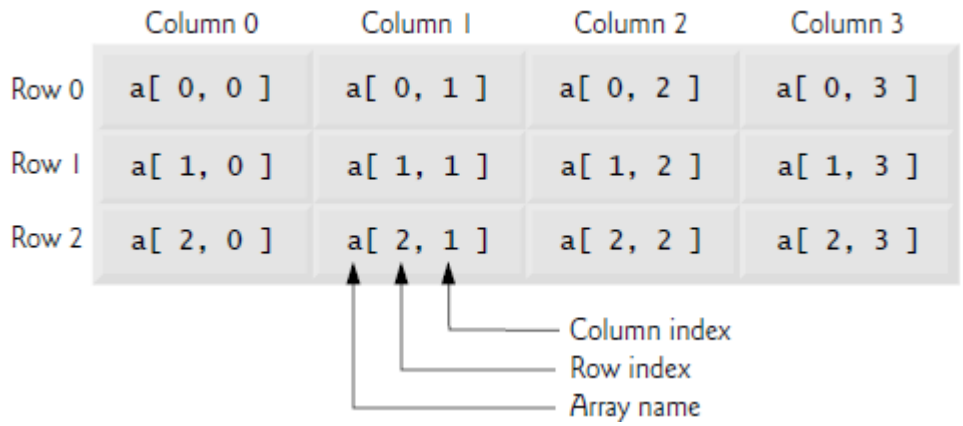
تُستخدم المصفوفات الثنائية لتخزين جدول من القيم يتألف من مجموعة من لأسطر ومجموعة من لأعمدة. تتحدّد لخلية في هذه لحالة بفهرسين: فهرس السطر وفهرس العمود.

تدعم لغة C# نوعين من المصفوفات الثنائية:

1. المصفوفات المستطيلة.
2. المصفوفات المسننة.

1. المصفوفات المستطيلة Rectangular Arrays:

- تتألف المصفوفة المستطيلة من مجموعة من الأسطر والأعمدة وبحيث يكون لكل صف نفس العدد من الأعمدة.
- يُبين الشكل التالي مصفوفة مستطيلة من ثلاثة أسطر وأربعة أعمدة.



- يتم الوصول لعنصر من المصفوفة المستطيلة باستخدام الشكل:

a[row, column]

حيث a هو اسم المصفوفة و row هو فهرس السطر و column هو فهرس العمود.

- يُمكن التصريح وإعطاء قيم ابتدائية للمصفوفة كما في لمثال التالي:

```
int[ , ] b = { { 1, 2 }, { 3, 4 } };
```

- كما يُمكن التصريح وإنشاء لمصفوفة مع تحديد عدد الأسطر وعدد لأعمدة:

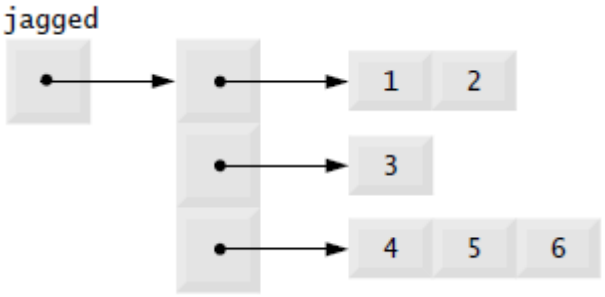
```
int[ , ] b;
b = new int[ 3, 4 ];
```

حيث يتم إنشاء مصفوفة من ثلاثة أسطر وأربعة أعمدة.

2. المصفوفات المسننة Jagged Arrays:

- تتكون المصفوفة المسننة من مصفوفة أحادية يؤشر كل عنصر منها على مصفوفة أحادية. تسمح هذه البنية بالحصول على مصفوفة ثنائية وبحيث يُمكن لكل سطر أن يكون له عدد مختلف من الأعمدة.
- يُمكن التصريح وإعطاء قيم ابتدائية للمصفوفة كما في المثال التالي:

```
int[][] jagged = { new int[] { 1, 2 },
new int[] { 3 },
new int[] { 4, 5, 6 } };
```



- كما يُمكن التصريح عن المصفوفة المسننة ومن ثم إنشاء أسطرها ثم إنشاء أعمدة مختلفة لكل سطر:

```
int[][] c;
c = new int[ 2 ][ ]; // create 2 rows
c[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
c[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

مثال: نقوم في المثال التالي بالتصريح عن مصفوفة مستطيلة وعن مصفوفة مسننة وإعطاء قيم ابتدائية لهما.

- نصرح عن نسخة أولى من الطريقة `OutputArray` والتي يكون لها مصفوفة مستطيلة كمعامل دخل:
  1. لاحظ استخدام الأقواس [,] مع المصفوفة المستطيلة.
  2. لاحظ استخدام الطريقة `GetLength(0)` لمعرفة البعد الأول للمصفوفة (عدد الأسطر).
  3. لاحظ استخدام الطريقة `GetLength(1)` لمعرفة البعد الثاني للمصفوفة (عدد الأعمدة).

- نصرح عن نسخة ثانية من الطريقة `OutputArray` والتي يكون لها مصفوفة مسننة كمعامل دخل:
  1. لاحظ استخدام الأقواس [][] مع المصفوفة المسننة.
  2. لاحظ استخدام التعليمة foreach للوصول إلى كل سطر في المصفوفة.
  3. لاحظ استخدام التعليمة foreach للوصول إلى كل عنصر في سطر.

- نُصرح أيضاً عن نسخة ثانية من الطريقة `OutputArray` والتي يكون لها معامل دخل مصفوفة مسننة:
  1. لاحظ استخدام الأقواس [][] مع المصفوفة المسننة.

2. لاحظ استخدام التعليمة foreach للوصول إلى كل سطر في المصفوفة.

3. لاحظ استخدام التعليمة foreach للوصول إلى كل عنصر في السطر.

```
// InitArray.cs
// Initializing rectangular and jagged arrays.
using System;

public class InitArray
{
   // create and output rectangular and jagged arrays
   public static void Main( string[] args )
   {
      // with rectangular arrays,
      // every row must be the same length.
      int[ , ] rectangular = { { 1, 2, 3 }, { 4, 5, 6 } };

      // with jagged arrays,
      // we need to use "new int[]" for every row,
      // but every row does not need to be the same length.
      int[][] jagged = { new int[] { 1, 2 },
                         new int[] { 3 },
                         new int[] { 4, 5, 6 } };

      OutputArray( rectangular ); // displays array rectangular by row
      Console.WriteLine(); // output a blank line
      OutputArray( jagged ); // displays array jagged by row
   } // end Main

   // output rows and columns of a rectangular array
   public static void OutputArray( int[ , ] array )
   {
      Console.WriteLine( "Values in the rectangular array by row are" );

      // loop through array's rows
      for ( int row = 0; row < array.GetLength( 0 ); ++row )
      {
         // loop through columns of current row
         for ( int column = 0; column < array.GetLength( 1 ); ++column )
            Console.Write( "{0}  ", array[ row, column ] );

         Console.WriteLine(); // start new line of output
      } // end outer for
   } // end method OutputArray

   // output rows and columns of a jagged array
   public static void OutputArray( int[][] array )
   {
      Console.WriteLine( "Values in the jagged array by row are" );

      // loop through each row
      foreach ( int[] row in array )
      {
         // loop through each element in current row
         foreach ( int element in row )
            Console.Write( "{0}  ", element );

         Console.WriteLine(); // start new line of output
      } // end outer foreach
   } // end method OutputArray
} // end class InitArray
```

- يكون ناتج التنفيذ:

```
Values in the rectangular array by row are
1  2  3
4  5  6

Values in the jagged array by row are
1  2
3
4  5  6
Press any key to continue . . .
```

2. مثال تعليمي

الأهداف التعليمية:

مثال تعليمي على استخدام المصفوفات الثنائية.

مثال تعليمي:

نُخزّن في المثال التالي نتائج عشرة طلاب في ثلاثة امتحانات وذلك باستخدام مصفوفة مستطيلة.

- تقوم الطريقة OutputGrades بطباعة عناصر مصفوفة مستطيلة (معامل الطريقة مصفوفة مستطيلة)
- تقوم الطريقة GetMinimum بإيجاد أصغر قيمة في مصفوفة مستطيلة (معامل الطريقة مصفوفة مستطيلة)
- تقوم الطريقة GetMaximum بإيجاد أكبر قيمة في مصفوفة مستطيلة (معامل الطريقة مصفوفة مستطيلة)
- تقوم الطريقة GetAverage بإيجاد وسطي علامات طالب (للطريقة معاملين: الأول فهرس الطالب والثاني مصفوفة مستطيلة)
- تقوم الطريقة OutputBarChart بعدّ العلامات في 11 مجال (العلامات المساوية إلى 100، العلامات بين 90 إلى 99، . . . ، العلامات بين 0 و9) ومن ثم إظهار عدد من النجوم لكل مجال وبحيث يكون عدد النجوم مساوي للعلامات المحصورة ضمن المجال
- تقوم الطريقة ProcessGrades باستدعاء كل من OutputGrades و OutputBarChart

```
// GradeBookTest.cs
// Create GradeBook object using a rectangular array of grades.
using System;
public class GradeBookTest
{
   static void ProcessGrades(int[,] grades)
    {
        // output grades array
        OutputGrades(grades);

        // call methods GetMinimum and GetMaximum
        Console.WriteLine("\n{0} {1}\n{2} {3}\n",
           "Lowest grade in the grade book is", GetMinimum(grades),
           "Highest grade in the grade book is", GetMaximum(grades));

        // output grade distribution chart of all grades on all tests
        OutputBarChart(grades);
    } // end method ProcessGrades

    // find minimum grade
   static int GetMinimum(int[,] grades)
    {
        // assume first element of grades array is smallest
        int lowGrade = grades[0, 0];

        // loop through elements of rectangular grades array
        foreach (int grade in grades)
        {
            // if grade less than lowGrade, assign it to lowGrade
            if (grade < lowGrade)
```

```
                lowGrade = grade;
        } // end foreach

        return lowGrade; // return lowest grade
    } // end method GetMinimum

    // find maximum grade
    static int GetMaximum(int[,] grades)
    {
        // assume first element of grades array is largest
        int highGrade = grades[0, 0];

        // loop through elements of rectangular grades array
        foreach (int grade in grades)
        {
            // if grade greater than highGrade, assign it to highGrade
            if (grade > highGrade)
                highGrade = grade;
        } // end foreach

        return highGrade; // return highest grade
    } // end method GetMaximum

    // determine average grade for particular student
    static double GetAverage(int student, int[,] grades)
    {
        // get the number of grades per student
        int amount = grades.GetLength(1);
        int total = 0; // initialize total

        // sum grades for one student
        for (int exam = 0; exam < amount; ++exam)
            total += grades[student, exam];

        // return average of grades
        return (double)total / amount;
    } // end method GetAverage

    // output bar chart displaying overall grade distribution
    static void OutputBarChart(int[,] grades)
    {
        Console.WriteLine("Overall grade distribution:");

        // stores frequency of grades in each range of 10 grades
        int[] frequency = new int[11];

        // for each grade in GradeBook, increment the appropriate frequency
        foreach (int grade in grades)
        {
            ++frequency[grade / 10];
        } // end foreach

        // for each grade frequency, display bar in chart
        for (int count = 0; count < frequency.Length; ++count)
        {
            // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
            if (count == 10)
                Console.Write("  100: ");
            else
                Console.Write("{0:D2}-{1:D2}: ",
                   count * 10, count * 10 + 9);
```

```
            // display bar of asterisks
            for (int stars = 0; stars < frequency[count]; ++stars)
                Console.Write("*");

            Console.WriteLine(); // start a new line of output
        } // end outer for
    } // end method OutputBarChart

    // output the contents of the grades array
    static void OutputGrades(int[,] grades)
    {
        Console.WriteLine("The grades are:\n");
        Console.Write("            "); // align column heads

        // create a column heading for each of the tests
        for (int test = 0; test < grades.GetLength(1); ++test)
            Console.Write("Test {0}  ", test + 1);

        Console.WriteLine("Average"); // student average column heading

        // create rows/columns of text representing array grades
        for (int student = 0; student < grades.GetLength(0); ++student)
        {
            Console.Write("Student {0,2}", student + 1);

            // output student's grades
            for (int grade = 0; grade < grades.GetLength(1); ++grade)
                Console.Write("{0,8}", grades[student, grade]);

            // call method GetAverage to calculate student's average grade;
            // pass row number as the argument to GetAverage
            Console.WriteLine("{0,9:F}", GetAverage(student, grades));
        } // end outer for
    } // end method OutputGrades

    // Main method begins application execution
    public static void Main( string[] args )
    {
       // rectangular array of student grades
       int[ , ] gradesArray = { { 87, 96, 70 },
                                { 68, 87, 90 },
                                { 94, 100, 90 },
                                { 100, 81, 82 },
                                { 83, 65, 85 },
                                { 78, 87, 65 },
                                { 85, 75, 83 },
                                { 91, 94, 100 },
                                { 76, 72, 84 },
                                { 87, 93, 73 } };


       ProcessGrades(gradesArray);
    } // end Main
} // end class GradeBookTest
```

- يكون ناتج التنفيذ:

```
Welcome to the grade book for
CS101 Introduction to C# Programming!

The grades are:

            Test 1  Test 2  Test 3  Average
Student  1      87      96      70    84.33
Student  2      68      87      90    81.67
Student  3      94     100      90    94.67
Student  4     100      81      82    87.67
Student  5      83      65      85    77.67
Student  6      78      87      65    76.67
Student  7      85      75      83    81.00
Student  8      91      94     100    95.00
Student  9      76      72      84    77.33
Student 10      87      93      73    84.33

Lowest grade in the grade book is 65
Highest grade in the grade book is 100

Overall grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: ******
80-89: **********
90-99: *******
  100: ***
Press any key to continue . . .
```

3. قائمة المعاملات متغيرة الطول

الأهداف التعليمية:

قائمة المعاملات متغيرة الطول.

قائمة المعاملات متغيرة الطول:

تسمح قائمة المعاملات متغيرة الطول بالتصريح عن طرق يُمكن أن يكون لها عدد متغير من المعاملات.

يتم التصريح عن قائمة من المعاملات متغير الطول باستخدام مصفوفة أحادية كمعامل مسبوقة بالكلمة المفتاحية params.

مثال: نُصرّح في المثال التالي عن الطريقة Average مع استخدام مصفوفة أحادية كمعامل دخل مع الكلمة المفتاحية params. لاحظ أننا نقوم باستدعاء هذه الطريقة مع عدد متغير من المعاملات في كل مرة.

```
// ParamArrayTest.cs
// Using variable-length argument lists.
using System;

public class ParamArrayTest
{
   // calculate average
   public static double Average( params double[] numbers )
   {
      double total = 0.0; // initialize total

      // calculate total using the foreach statement
      foreach ( double d in numbers )
         total += d;

      return total / numbers.Length;
   } // end method Average

   public static void Main( string[] args )
   {
      double d1 = 10.0;
      double d2 = 20.0;
      double d3 = 30.0;
      double d4 = 40.0;

      Console.WriteLine(
         "d1 = {0:F1}\nd2 = {1:F1}\nd3 = {2:F1}\nd4 = {3:F1}\n",
         d1, d2, d3, d4 );

      Console.WriteLine( "Average of d1 and d2 is {0:F1}",
         Average( d1, d2 ) );
      Console.WriteLine( "Average of d1, d2 and d3 is {0:F1}",
         Average( d1, d2, d3 ) );
      Console.WriteLine( "Average of d1, d2, d3 and d4 is {0:F1}",
         Average( d1, d2, d3, d4 ) );
   } // end Main
} // end class ParamArrayTest
```

- يكون ناتج التنفيذ:

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
Press any key to continue . . .
```

4. التمارين

تمرين:

يوجد في شركة مبيعات 3 مندوبي مبيعات يقومون ببيع 5 سلع.

يتم إدخال مبلغ مبيعات كل موظف من كل سلعة في مصفوفة ثنائية.

قم بكتابة تطبيق يحوي الطرق التالية:

- طريقة نمرر لها مصفوفة المبيعات فتقوم بطلب قيم المبيعات وتخزينها في المصفوفة
- طريقة نمرر لها مصفوفة المبيعات فتقوم بإظهار مجموع مبيعات كل موظف
- طريقة نمرر لها مصفوفة المبيعات فتقوم بإظهار مجموع المبيعات من كل سلعة

Exceptions الاستثناءات

الكلمات المفتاحية:

الاستثناءات، التقاط ومعالجة الاستثناءات، try, catch, finally.

ملخص:

نستعرض في هذا الفصل كيفية معالجة الاستثناءات التي يُمكن أن تظهر خلال تنفيذ البرامج.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- معالجة الاستثناءات
- التعليمة try, catch, finally

المخطط:

يتضمن فصل "الاستثناءات Exceptions" وحدة عنوانها:

1. الاستثناءات.

1. الاستثناءات

الأهداف التعليمية:

التقاط الاستثناءات.

مقدمة:

- تُطلق الاستثناءات عند حصول مشكلة غير متوقعة أثناء تنفيذ البرنامج
- تسمح معالجة الاستثناءات بكتابة تطبيقات تستمر في عملها بعد حدوث الاستثناءات مما يجعل هذه التطبيقات أكثر مرونة

يُبين المثال التالي السلوك الافتراضي في حال عدم التقاط ومعالجة الاستثناءات

يُمكن في المثال التالي حصول استثناء من النوع قسمة على صفر في حال قيام المستخدم بإدخال قيمة الصفر للعدد المقسوم عليه

```
// DivideByZeroNoExceptionHandling.cs
// Integer division without exception handling.
using System;

class DivideByZeroNoExceptionHandling
{
   static void Main()
   {
      // get numerator
      Console.Write( "Please enter an integer numerator: " );
      int numerator = Convert.ToInt32( Console.ReadLine() );

      // get denominator
      Console.Write( "Please enter an integer denominator: " );
      int denominator = Convert.ToInt32( Console.ReadLine() );

      // divide the two integers, then display the result
      int result = numerator / denominator;
      Console.WriteLine( "\nResult: {0:D} / {1:D} = {2:D}",
         numerator, denominator, result );
   } // end Main
} // end class DivideByZeroNoExceptionHandling
```

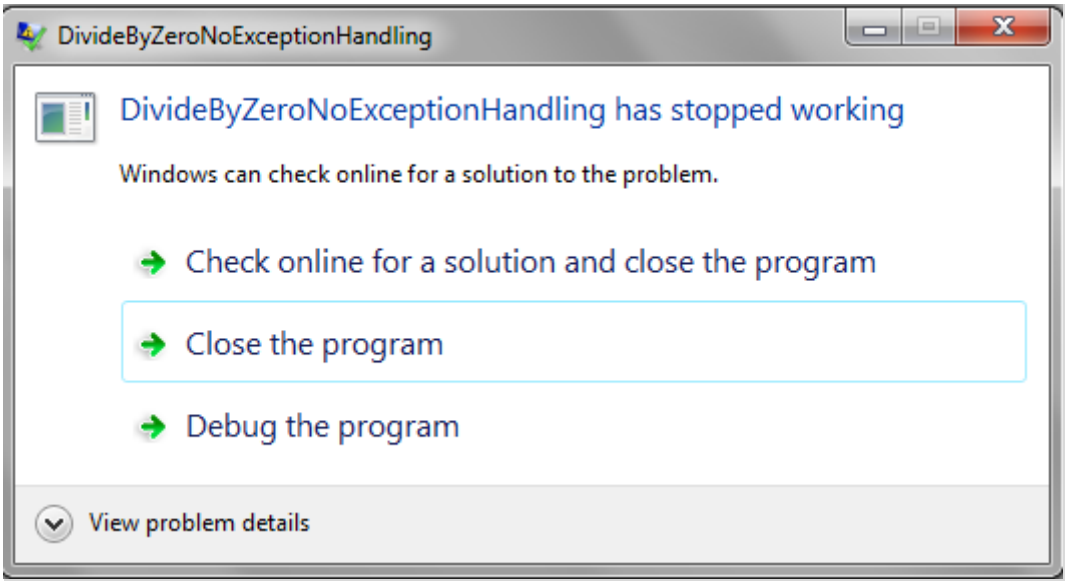
- يُعطي تنفيذ البرنامج مثلاً:

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

أما في حال قيام المستخدم بإدخال قيمة مساوية للصفر للعدد المقسوم عليه، فسيتم إظهار الاستثناء:

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Unhandled Exception: System.DivideByZeroException: Attempted to divide by
zero.
   at DivideByZeroNoExceptionHandling.Main() in
e:\5.OOP\Chapter_8\DivideByZeroN
oExceptionHandling\DivideByZeroNoExceptionHandling\DivideByZeroNoExc
eptionHandli
ng.cs:line 18
```

- كما يتوقف تنفيذ البرنامج وتظهر النافذة التالية:



التقاط الاستثناءات:

تسمح التعليمة try, catch, finally بإلتقاط الاستثناءات حين وقوعها ومن ثم تنفيذ تعليمات محدّدة عوضاً عن توقف البرنامج.

يكون للتعليمة الشكل العام التالي:

```
        try
        {
            //Execution starts here
            // This code may or may not throw an exception
        }
        catch
        {
            // if an exception is thrown in the try block,
            // code in this catch block is activated.
        }
        finally
        {
            // Code in this block executes when the routine
ends,
            // regardless of Whether an exception was
thrown.
        }
```

- حين وقوع أي استثناء في الكتلة try ينتقل التنفيذ إلى كتلة catch.
- يتم تنفيذ تعليمات الكتلة finally في جميع الحالات.
- يُمكن وضع أكثر من كتلة catch وبحيث تختص كل كتلة بنوع معين من الاستثناءات.
- يُبين المثال التالي التقاط استثناء القسمة على صفر DivideByZeroException واستثناء التنسيق FormatException.
- عند حصول أي مشكلة أثناء التنفيذ في كتلة `try`، يتم انتقال التحكم إلى أحد الكتلتين `catch` وذلك حسب نوع الاستثناء المُنطلق.

- يتم في الكتلة `catch` في المثال التالي إظهار رسالة الخطأ الموافقة للمستخدم. ويتم متابعة التنفيذ وبدون توقف البرنامج:

```
// DivideByZeroExceptionHandling.cs
// FormatException and DivideByZeroException handlers.
using System;

class DivideByZeroExceptionHandling
{
   static void Main( string[] args )
   {
      bool continueLoop = true; // determines whether to keep looping

      do
      {
         // retrieve user input and calculate quotient
         try
         {
            // Convert.ToInt32 generates FormatException
            // if argument cannot be converted to an integer
            Console.Write( "Enter an integer numerator: " );
            int numerator = Convert.ToInt32( Console.ReadLine() );
            Console.Write( "Enter an integer denominator: " );
            int denominator = Convert.ToInt32( Console.ReadLine() );

            // division generates DivideByZeroException
            // if denominator is 0
            int result = numerator / denominator;

            // display result
            Console.WriteLine( "\nResult: {0} / {1} = {2}",
               numerator, denominator, result );
            continueLoop = false;
         } // end try
         catch ( FormatException formatException )
         {
            Console.WriteLine( "\n" + formatException.Message );
            Console.WriteLine(
               "You must enter two integers. Please try again.\n" );
         } // end catch
         catch ( DivideByZeroException divideByZeroException )
         {
            Console.WriteLine( "\n" + divideByZeroException.Message );
            Console.WriteLine(
               "Zero is an invalid denominator. Please try again.\n" );
         } // end catch
      } while ( continueLoop ); // end do...while
   } // end Main
} // end class DivideByZeroExceptionHandling
```

- يُمكن أن يكون التنفيذ بدون استثناءات:

```
Enter an integer numerator: 100
Enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

في حال قيام المستخدم بإدخال قيمة الصفر للمقسوم عليه، سيتم إطلاق استثناء القسمة على صفر `DivideByZeroException` ومعالجته في الكتلة `catch` الموافقة:

```
Enter an integer numerator: 100
Enter an integer denominator: 0

Attempted to divide by zero.
Zero is an invalid denominator. Please try again.

Enter an integer numerator: 100
Enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

في حال قيام المستخدم مثلاً بإدخال قيمة نصية عوضاً عن قيمة رقمية للمقسوم عليه، سيتم إطلاق استثناء التنسيق `FormatException` ومعالجته في الكتلة `catch` الموافقة:

```
Enter an integer numerator: 100
Enter an integer denominator: Hello

Input string was not in a correct format.
You must enter two integers. Please try again.

Enter an integer numerator: 100
Enter an integer denominator: 7

Result: 100 / 7 = 14
Press any key to continue . . .
```

2. التمارين

تمرين:

قم بإعادة كتابة تطبيق حل معادلة من الدرجة الثانية:

$$A\,x^2 + B\,x + C = 0$$

يقوم التطبيق بسؤال المستخدم لإدخال القيم A, B, C.

بعدها يُظهر التطبيق جذور المعادلة إن وجدت أو عبارة "لاجذور حقيقية".

يجب أن يُظهر التطبيق رسائل مناسبة في حال إدخال المستخدم لقيم غير مقبولة (نصوص مثلاً عوضاً عن الأعداد). في حال وقوع أي مشكلة في الإدخال يعاود التطبيق سؤال المستخدم عن القيم لإدخالها من جديد.

السلاسل النصية

الكلمات المفتاحية:

طرق السلاسل النصية.

ملخص:

نستعرض في هذا الفصل أهم طرق التعامل مع السلاسل النصية.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- السلاسل النصية
- أهم الطرق المعرّفة على السلاسل النصية

المخطط:

يتضمن فصل "السلاسل النصية" وحدة عنوانها:

1. السلاسل النصية.

1. السلاسل النصية

الأهداف التعليمية:

التعامل مع السلاسل النصية.

نقوم في الأمثلة التالية بالتصريح عن سلاسل نصية:

```
string newString = "This is a string literal";
string literalOne = "\\\\MySystem\\MyDirectory\\ProgrammingC#.cs";
string verbatimLiteralOne = @"\\MySystem\MyDirectory\ProgrammingC#.cs";

string literalTwo = "Line One\nLine Two";
```

- لاحظ استخدام المحرف الخاص (\) للدلالة على أن المحرف التالي يجب أن يُعتبر كما هو وليس محرف خاص
- لاحظ استخدام المحرف (@) لإلغاء جميع المحارف الخاصة ضمن السلسلة
- لاحظ مثلاً أن (\n) تعني سطر جديد

الطريقة ( )ToString:

تُستخدم هذه الطريقة مع جميع الأنماط للتحويل إلى سلسلة نصية. مثلاً:

```
    int myInteger = 5;
    string integerString = myInteger.ToString();
```

أهم الطرق للتعامل مع السلاسل النصية:

يُبين الجدول التالي أهم الطرق المتوفرة للتعامل مع السلاسل النصية:

الطريقة	الشرح
Compare()	تُستخدم للمقارنة بين السلاسل. تكون القيمة المعادة: • 0 إذا تساوت السلسلتين • 1 إذا كانت السلسلة الأولى أكبر من الثانية • −1 إذا كانت السلسلة الثانية أصغر من الثانية تأخذ افتراضياً هذه الطريقة حالة الأحرف عند المقارنة. إذا أردنا تجاهل حالة الأحرف نضع القيمة true في المعامل الثاني.
Concat()	جمع سلسلتين نصيتين.
Copy()	إنشاء نسخة من سلسلة.
Equals()	تُستخدم للمقارنة بين السلاسل: تُعيد true إذا تساوت السلسلتين.
Length	عدد محارف السلسلة النصية
EndsWith()	اختبار انتهاء السلسلة بسلسلة أخرى
Insert()	إدراج سلسلة ضمن السلسلة اعتباراً من فهرس معين
LastIndexOf()	فهرس آخر ظهور لسلسلة
StartsWith()	اختبار ابتداء السلسلة بسلسلة أخرى
Substring()	الحصول على سلسلة جزئية من السلسلة
ToLower()	التحويل لأحرف صغيرة
ToUpper()	التحويل لأحرف كبيرة
Trim()	حذف الفراغات الزائدة من بداية ونهاية السلسلة
TrimEnd()	حذف الفراغات الزائدة من نهاية السلسلة
TrimStart()	حذف الفراغات الزائدة من بداية السلسلة

مثال تعليمي1:

يُبين المثال التالي استخدام بعض طرق السلاسل النصية:

```
using System;
class Strings
{
static void Main( string[] args )
{
// create some strings to work with
string s1 = "abcd";
string s2 = "ABCD";
string s3 = @"Liberty Associates, Inc.
              provides custom .NET development,
              on-site Training and Consulting";

int result;  // hold the results of comparisons
// compare two strings, case sensitive
result = string.Compare(s1, s2);
Console.WriteLine("compare s1: {0}, s2: {1}, result: {2}\n", s1, s2, result);

result = string.Compare("Said", "Ahmad Said");
Console.WriteLine("compare Said, Ahmad Said, result: {2}\n", s1, s2, result);

// overloaded compare, takes boolean "ignore case"
//(true = ignore case)
result = string.Compare(s1,s2, true);
Console.WriteLine("compare insensitive\n");
Console.WriteLine("s4: {0}, s2: {1}, result: {2}\n", s1, s2, result);
// concatenation method
string s6 = string.Concat(s1,s2);
Console.WriteLine( "s6 concatenated from s1 and s2: {0}", s6);

// use the overloaded operator
string s7 = s1 + s2;
Console.WriteLine("s7 concatenated from s1 + s2: {0}", s7);

// the string copy method
string s8 = string.Copy(s7);
Console.WriteLine("s8 copied from s7: {0}", s8);

// use the overloaded operator
string s9 = s8;
Console.WriteLine("s9 = s8: {0}", s9);

// three ways to compare.
Console.WriteLine("\nDoes s9.Equals(s8)?: {0}", s9.Equals(s8));

Console.WriteLine( "Does Equals(s9,s8)?: {0}", string.Equals(s9,s8));

Console.WriteLine("Does s9==s8?: {0}", s9 == s8);

// Tow useful properties: the index and the length
Console.WriteLine( "\nString s9 is {0} characters long. ",  s9.Length);

Console.WriteLine("The 5th character is {0}\n", s9[4]);

// test whether a string ends with a set of characters
Console.WriteLine("s3:{0}\nEnds with Training?: {1}\n",s3, s3.EndsWith("Training") );
Console.WriteLine("Ends with Consulting?: {0}",s3.EndsWith("Consulting"));
```

```
// return the index of the substring
Console.WriteLine("\nThe first occurrence of Training ");
Console.WriteLine ("in s3 is {0}\n", s3.IndexOf("Training"));
// insert the word excellent before "training"
string s10 = s3.Insert(101,"excellent ");
Console.WriteLine("s10: {0}\n",s10);
// you can combine the two as follows:
string s11 = s3.Insert(s3.IndexOf("Training"), "excellent ");
Console.WriteLine("s11: {0}\n",s11);
}
}
```

يكون ناتج التنفيذ:

```
compare s1: abcd, s2: ABCD, result: -1

compare Said, Ahmad Said, result: 1

compare insensitive

s4: abcd, s2: ABCD, result: 0

s6 concatenated from s1 and s2: abcdABCD
s7 concatenated from s1 + s2: abcdABCD
s8 copied from s7: abcdABCD
s9 = s8: abcdABCD

Does s9.Equals(s8)?: True
Does Equals(s9,s8)?: True
Does s9==s8?: True

String s9 is 8 characters long.
The 5th character is A

s3:Liberty Associates, Inc.
           provides custom .NET development,
           on-site Training and Consulting
Ends with Training?: False

Ends with Consulting?: True

The first occurrence of Training
in s3 is 99

s10: Liberty Associates, Inc.
           provides custom .NET development,
           on-site Trexcellent aining and Consulting

s11: Liberty Associates, Inc.
           provides custom .NET development,
           on-site excellent Training and Consulting

Press any key to continue . . .
```

مثال تعليمي2:

يُبين المثال التالي طرق البحث ضمن السلاسل النصية:

```
using System;
class Strings
{
static void Main( string[] args )
{
    string s1 = "One Two Three Four";
    int ix;
    // get the index of the last space
    ix = s1.LastIndexOf(" ");

    // get the last word.
    string s2 = s1.Substring(ix + 1);

    // set s1 to the substring starting at 0
    // and ending at ix (the start of the last word
    // thus s1 has one two three
    s1 = s1.Substring(0, ix);

    // find the last space in s1 (after two)
    ix = s1.LastIndexOf(" ");
    // set s3 to the substring starting at
    // ix, the space after "two" plus one more
    // thus s3 = "three"
    string s3 = s1.Substring(ix + 1);

    Console.WriteLine("s2: {0}\ns3: {1}", s2, s3);
    // reset s1 to the substring starting at 0
    // and ending at ix, thus the string "one two"
    s1 = s1.Substring(0, ix);

    // reset ix to the space between "one" and "two"
    ix = s1.LastIndexOf(" ");

    // set s4 to the substring starting one space after ix, thus the substring "two"
    string s4 = s1.Substring(ix + 1);

    // reset s1 to the substring starting at 0 and ending at ix, thus "one"
    s1 = s1.Substring(0, ix);

    // set ix to the last space, but there is  none so ix now = -1
    ix = s1.LastIndexOf(" ");

    // set s5 to the substring at one past the last space.
    // there was no last space  so this sets s5 to the substring starting at zero
    string s5 = s1.Substring(ix + 1);

    Console.WriteLine("s4: {0}\ns5: {1}\n", s4, s5);
    Console.WriteLine("s1: {0}\n", s1);
}
}
```

- يكون ناتج التنفيذ:

```
s2: Four
s3: Three
s4: Two
s5: One

s1: One

Press any key to continue . . .
```

2. التمارين

تمرين:

قم بكتابة طريقة نُمرر لها سلسلة نصية فتقوم بتحويل الحرف الأول من كل كلمة إلى حرف كبير. مثلاً من أجل الدخل:

I go to school every day

يكون الخرج:

I Go To School Every Day

التعامل مع الملفات

الكلمات المفتاحية:

الملفات التسلسلية، الملفات ذات الوصول العشوائي.

ملخص:

نستعرض في هذا الصف أنواع وطرق التعامل مع الملفات.

الأهداف التعليمية:

يتعرف الطالب في هذا الفصل على:

- استخدام الملفات التسلسلية
- استخدام الملفات ذات الوصول العشوائي

المخطط:

يتضمن فصل "التعامل مع الملفات" وحدتين عناوينها بالترتيب المحدد:

1. الملفات التسلسلية.
2. الملفات ذات الوصول العشوائي.

1. الملفات التسلسلية

الأهداف التعليمية:

التعامل مع الملفات التسلسلية.

الملفات التسلسلية:

يُمكنك في هذا النوع من الملفات قراءة أو كتابة البيانات من بداية الملف إلى نهايته.

الكتابة في ملف تسلسلي:

- تحتاج للكتابة في ملف تسلسلي إلى كائنين:
  1. الكائن الأول من النمط `FileStream`.
  2. الكائن الثاني من النمط `StreamWriter`.
- يقبل باني الصف `StreamWriter` معامل من النمط `FileStream`.
- يكون لباني الصف `FileStream` ثلاثة معاملات:
  1. مسار الملف.
  2. نمط فتح الملف.
  3. نمط الوصول.
- يُمكن أن يكون نمط فتح الملف أحد الأنواع الأربعة التالية:
  1. Append: يتم إضافة النص إلى نهاية الملف إن كان الملف موجوداً. وإلا يتم إنشاء ملف جديد.
  2. Create: يتم إنشاء ملف جديد. في حال كان الملف موجود مسبقاً يتم الكتابة فوقه.
  3. CreateNew: يتم إنشاء ملف جديد. في حال كان الملف موجود مسبقاً يتم إطلاق خطأ.
  4. OpenorCreate: فتح الملف إن كان موجوداً. إذا لم يكن موجوداً يتم إنشاءه.
- يُمكن أن يكون نمط الوصول أحد الأنواع الثلاثة التالية:
  1. Read : يُمكن فقط القراءة من الملف.
  2. ReadWrite : يُمكن القراءة والكتابة.
  3. Write : يُمكن فقط الكتابة.

القراءة من ملف تسلسلي:

- تحتاج للقراءة من ملف تسلسلي إلى كائنين:
  1. الكائن الأول من النمط `FileStream`.
  2. الكائن الثاني من النمط `StreamReader`.
- يقبل باني الصف `StreamReader` معامل من النمط `FileStream`.

مثال: يُبين المثال التالي استخدام الملفات التسلسلية:

```
using System;
using System.IO;

namespace SequentialDemo
{
    class Program
    {
        static void writeObject(string path)
        {
            FileStream fs;
            StreamWriter fw;
            try
            {
             //create file stream object
            fs = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write);
            //create writer object
            fw = new StreamWriter(fs);
            //write text to file
            fw.WriteLine("C# Programming");
            fw.WriteLine("C Programming");
            fw.WriteLine("C++ Programming");
            fw.Close();
            fs.Close();
            }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
        static void readObject(string path)
        {
            FileStream fs;
            StreamReader fr;
            try
            {
                //create file stream object
                fs = new FileStream(path, FileMode.Open, FileAccess.Read);
                //create reader objec
                fr = new StreamReader(fs);
                string content;
                while (!fr.EndOfStream)
                {
                    content = fr.ReadLine();
                    Console.WriteLine(content);
                }
                fr.Close();
                fs.Close();
            }
            catch (Exception e) { Console.WriteLine(e.Message); }
        }
        static void Main(string[] args)
        {
            writeObject("d:\\students.txt");
            readObject("d:\\students.txt");
            Console.Read();
        }
    }
}
```

- يكون ناتج التنفيذ:

```
C# Programming
C Programming
C++ Programming
```

2. الملفات ذات الوصول العشوائي

الأهداف التعليمية:

الملفات ذات الوصول العشوائي.

الملفات ذات الوصول العشوائي:

- يُمكن الوصول في هذا النوع من الملفات لأي تسجيلة مباشرة وبسرعة أي دون المرور بعدد كبير من التسجيلات
- للكتابة في ملف ذو وصول عشوائي، نستخدم الصف `BinaryWriter`
- للقراءة من ملف ذو وصل عشوائي، نستخدم الصف `BinaryReader`
- يجب تحديد مكان الكتابة أو القراءة باستخدام الخاصية `Position`
- نقوم في المثال التالي بتخزين بعض البيانات عن الطلاب ومن ثم قراءتها
- يحوي صف الطالب `Student` ثلاثة خصائص: سلسلة نصية لرقم الطالب `stnumber` وسلسلة نصية لاسم الطالب `stname` والحجم الأعظمي لتسجيلة الطالب `recordsize`
- تقوم الطريقة `addrecord` بتكرار الطلب من المستخدم إدخال رقم واسم الطالب ومن ثم إنشاء تسجيلة من النمط `Student` ومن ثم كتابتها في الملف
- نستخدم المتغير الساكن `pos` للحفاظ على عدد التسجيلات

مثال: تخزين بعض البيانات عن الطلاب ومن ثم قراءتها.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace RandomDemo
{
    class Program
    {
        public static int pos = 0;
        static void Main(string[] args)
        {
            addrecord("D:\\student.txt");
            readFromFile("D:\\student.txt");
        }

        class Student
        {
            private string stnumber;
            private string stname;
            private int recordsize;
            public string stunumber
            { //stunumber property

                set { stnumber = value; }
                get { return stnumber; }
            }
            public string stuname
            { //stuname property
```

```
                set { stname = value; }
                get { return stname; }

            }
            public int size
            {
                get { return calsize(); }
            }
            private int calsize()
            {
                recordsize = 2 * 15 + 2 * 20;// max record size
                return recordsize;
            }
        }
static void addrecord(string path)
        {
            Student stu = new Student();
            String con = "y";
            while (con != "n")
            {
                Console.Write("Enter student number:");
                stu.stunumber = Console.ReadLine();
                Console.Write("Enter student name:");
                stu.stuname = Console.ReadLine();
                pos += 1; //update position
                writeToFile(path, stu, pos, stu.size);
                Console.WriteLine("Continue?y/n:");
                con = Console.ReadLine();
            }
        }

static void writeToFile(string filename, Student obj, int pos, int size)
        {
            FileStream fout;
            BinaryWriter bw;
            //create a file stream object
            fout = new FileStream(filename, FileMode.Append, FileAccess.Write);
            //create a binary writer object
            bw = new BinaryWriter(fout);
            //set file position where to write data
            fout.Position = pos * size;
            //write data
            bw.Write(obj.stunumber);
            bw.Write(obj.stuname);
            //close objects
            bw.Close();
            fout.Close();
        }

static void readFromFile(string filename)
        {
            FileStream fn;
            BinaryReader br;
            Student stu = new Student();
            int currentrecord = 0;
            //open file to read data
            fn = new FileStream(filename, FileMode.Open, FileAccess.Read);
            br = new BinaryReader(fn);
            //read next record
            int i;
            for (i = 1; i <= (int)(fn.Length) / stu.size; i++)
            {
```

```
                currentrecord += 1; //update currentrecord position
                fn.Seek(currentrecord * stu.size, 0);
                stu.stunumber = br.ReadString().ToString();
                stu.stuname = br.ReadString().ToString();
                Console.WriteLine(stu.stunumber + "\t" + stu.stuname);
            }
            //update pos to the current position
            pos = currentrecord;
            //close objects
            br.Close();
            fn.Close();
        }
    }
}
```

- يكون ناتج التنفيذ مثلاً:

```
Enter student number:1
Enter student name:lolo
Continue?y/n:
y
Enter student number:2
Enter student name:toto
Continue?y/n:
y
Enter student number:3
Enter student name:mimi
Continue?y/n:
n
1       lolo
2       toto
3       mimi
Press any key to continue . . .
```

3. التمارين

تمرين:

قم بكتابة تطبيق لإدارة دليل هاتفي يسمح بالعمليات الأساسية: إضافة، حذف، تعديل، حذف.
يتم تخزين الاسم ورقم الهاتف ورقم الجوال لكل شخص في الدليل.

- استخدم الملفات التسلسلية في نسخة أولى.
- استخدم الملفات ذات الوصول العشوائي في نسخة ثانية.