
# مقدمة في البرمجة
# Introduction To Programming

**IPG101**

## الفصل السادس

## بنى التحكم ( الاختيار أو القرار)

## Control Structure (Selection or Decision)

## الكلمات المفتاحية

تدفق، تحكم، مجال رؤية، تعبير بولياني، شرط، أمر شرطي، if، else، switch.

## ملخص الفصل

يستعرض هذا الفصل مفهوم التحكم في مسار تنفيذ البرنامج المكتوب بلغة C#، حيث يركز على عمليات الاختيار أو إتخاذ القرار، ويركز على استعراض أساليب استخدام الأوامر الثلاثة للاختيار if، switch والمعامل الثلاثي. كما يلقي الضوء على بعض المفاهيم الهامة والمساعدة في هذا الأمر كمفهوم كتل التعليمات ومجال رؤية المتحولات.

## أهداف الفصل

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

- فهم أسلوب التحكم بالتدفق في لغة C#.
- التمييز بين الأوامر البسيطة والأوامر المركبة.
- فهم مجال رؤية المتحول.
- معرفة مبدأ عمل أمري الاختيار if و switch.
- استخدام الأشكال الثلاثة لأمر الاختيار if في حل المسائل.
- استخدام أمر الاختيار switch في حل مسألة ذات خيارات متعددة.

## محتويات الفصل

1. مقدمة
2. مفهوم التحكم في تنفيذ البرنامج.
3. كتل التعليمات ومجالات الرؤية.
4. الأمر الشرطي if.
5. أمر الاختيار المتعدد switch.
6. المعامل الثلاثي
7. تمارين وأنشطة.

## 1- مقدمة.

أمكن لنا مما سبق القول أن البرنامج هو عبارة مجموعة من الأوامر والتعابير البرمجية المكتوبة وفق ترتيب يؤدي إلى حل المسألة.

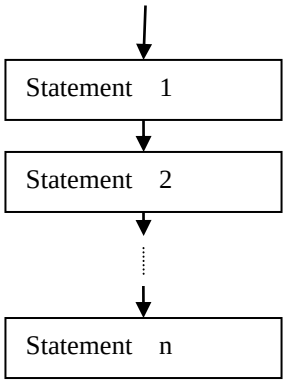
يمثل الأمر statement أصغر وحدة بنيوية في بناء البرنامج، بمعنى آخر، يمثل كل أمر خطوة ذات فعل مؤثر ( يقصد بالفعل المؤثر تغيير في حالة البرنامج، كتغيير قيمة متحول نتيجة عملية إسناد على سبيل المثال ).
إن تجميع هذه الأوامر يمكن البرنامج من تحقيق وظيفة محددة.

إن الترتيب الذي تنفذ بموجبه هذه الأوامر يدعى تدفق التحكم flow control، بمعنى أن، التعليمة التي تنفذ حالياً هي التي تملك التحكم بالبرنامج وعند انتهائها سيتم نقل التحكم transfer of control إلى التعليمة التالية.

قياسياً، يكون تدفق التحكم تسلسلياً sequential ( أي من تعليمة إلى التالية )، إلا أننا قد نصادف حالات يتفرع فيها التحكم إلى مسارات أخرى من خلال أوامر تفرع branch statements.
إن تدفق التحكم أمر هام لأنه يحدد ما الذي سينفذ خلال عملية تنفيذ البرنامج وما الذي لن ينفذ وهذا ما يؤثر على الناتج الإجمالي للبرنامج.

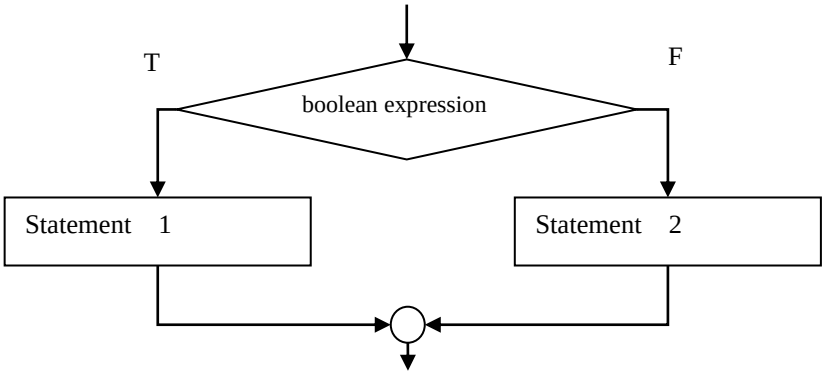
## 2- مفهوم التحكم في تنفيذ البرنامج

يعبر **التنفيذ التسلسلي** عن تنفيذ سلسلة من الأوامر بالترتيب الذي وردت فيه، بحيث يتم تنفيذ جميع الأوامر، وكل أمر ينفذ لمرة واحدة فقط.
يمكن تمثل هذا الأمر من خلال مخطط التدفق التالي:



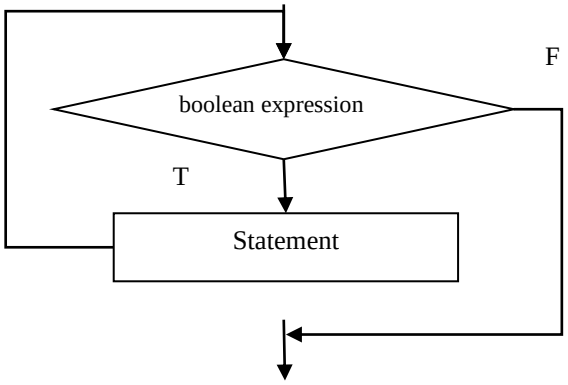

**الشكل 1- تدفق التنفيذ التسلسلي.**

إلا ان الأمر ليس دائماً على هذا النحو، فقد يتطلب تنفيذ برنامج أن لا يتم تنفيذ جميع الأوامر وإنما اختيار تنيفذ أوامر وتجاهل تنفيذ أوامر أخرى بحسب ناتج اختبار **شروط بوليانية** ما، كما يوضح المخطط التدفقي التالي:




**الشكل 2– تدفق التنفيذ الاختياري**

وبالمثل، قد يتطلب تنفيذ برنامج أن لا يكتفى بتنفيذ الأمر لمرة واحدة وإنما تكرار تنيفذه لعدد من المرات بحسب ناتج اختبار **شروط بوليانية** ما، كما يوضح المخطط التدفقي التالي:




**الشكل 3– تدفق التنفيذ التكراري**

**ملاحظة:** يقصد بالتعبير البولياني، تعبير بسيط أو مركب يعيد عند تقييمه قيمة بوليانية true أو false.

## 3– كتل التعليمات ومجالات الرؤية

تحتوي لغة C# عدة أشكال من الأوامر بما يحقق أغراضها المختلفة، حيث تنقسم هذه الأوامر إلى بسيطة ومركبة.

تستخدم **الأوامر البسيطة simple statements** من إنجاز إنجاز عملية واحدة، حيث يمثل الأمر البسيط عملية حسابية تنتهي بفاصلة منقوطة، كمثال:

```
int myValue;                أمر تصريح//
myValue++;                  // أمر إسناد، يغير حالة المتحول
double newValue=10.5; //أمر تصريح وإسناد، يغير حالة المتحول
```

يمكن أن يتم تجميع عدة أوامر بسيطة لتكوين **أمر مركب compound statement** من خلال وضعها بين قوسين { }، فمثلاً:

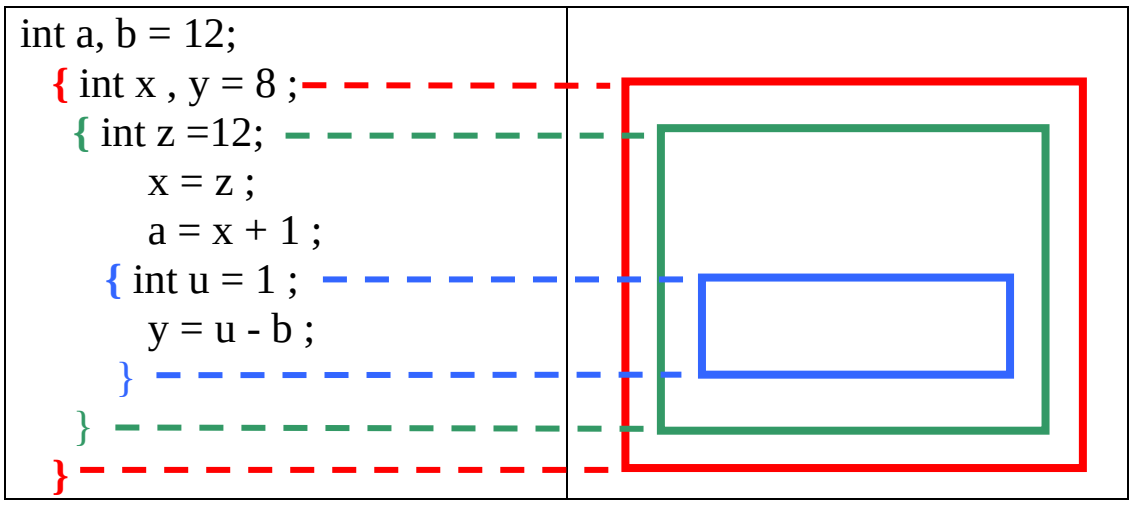
```
{
        int a=10,b=20,c;
        c=a+b;
        Console.WriteLine(c);
}
```

تأتي أهمية الأمر المركب من أنه يتيح استعمال عدة أوامر بمثابة أمر واحد ( وهذا ما سنلحظ أهميته في بنى التحكم )، إضافة إلى أنه يتيح تعريف مجال رؤية جديد scope ضمن البرنامج.

## مفهوم مجال الرؤية للمتحول Scope of Variable

يمكن للأمر المركب (كتلة التعليمات) أن تحتوي كتلة تعليمات أخرى على أن تكون **الكتل معلّبة ببعضها البعض تماماً**، إذ لايمكن أن تتقاطع كتلتي تعليمات جزئياً بأن تكون بداية الثانية بعد بداية الأولى وأن تكون نهاية الثانية بعد نهاية الأولى مثلاً.

**مثال:**




**الشكل 4– كتل تعليمات متداخلة**

نعرف مدى (مجال رؤية) المتحول بالكتلة التي يكون المتحول فيها مُعرَّفاً، ويكون المتحول مُعرّفاً في الكتلة التي تم تعريفه فيها وفي جميع الكتل المحتواة في كتلته، ولكنه لايكون مُعرّفاً في الكتل التي تحوي كتلته أو الكتل الموازية لكتلته.

**مثال:**

ليكن لدينا البرنامج التالي:

```
//Block0
int a, b = 12;

//Block1
{
        int x , y = 8 ;
}
// Block2
{
        int z =12;
    a = b + z;
        x = z ; //error
        a = x + 1 ;       //error
}
//Block3
{
        int u = 1 ;
    a = b - u;
        y = u - b ; //error
}
```

تظهر الأخطاء نتيجة عدم وجود أي تعريف لكل من x و y في الكتل التي تظهر بها، في حين لا توجد أي مشكلة في استخدام المتحولات a و b في هذه الكتل.

**ملاحظات هامة:**

في اللغات شبيهات C (C-like: C, C++, Java, C#) من المسموح تعريف المتحولات في أي كتلة، ومدى المتحول المصرح داخل كتلة يشمل كافة الكتل المحتواة فيها (كل الكتل الداخلية). يمكن أن يكون سطر تعريف المتحول في أي مكان في الكتلة، ولكن الممارسة البرمجية الأفضل هي في وضع تعريف متحولات الكتلة في بدايتها.

من حيث المبدأ يمكن أن نعِّرف متحولات بنفس الاسم في أي من الكتل المتداخلة. لأن ذلك يعني استقلالية المتحولات في كل كتلة (كل متحول، يعرَّف في كتلة، تُحجَز له ذاكرته الخاصة). لكن في C#، لا يمكن استخدام تعريف بنفس الاسم في الكتل المتداخلة، إلا في حالة كتلة الصف وكتلة البرنامج الجزئي/التابع (كما سنرى لاحقاً).

**تمرين:**

أين ستظهر الأخطاء الناجمة عن تعريف مدى المتحولات في البرنامج التالي:

```
int a, b = 12;
{
        int x , y = 8 ;
        {
                int k=0;
                x = a * y;
                k=k+z;
        }
}
{
        int z =12;
        a = z + b;
        {
                int u=0;
                x = a * y;
                u++;
        }
}
{
        int u = 1 ;
        a= u - b;
        z = u - b ;
}
```

**الحل:**

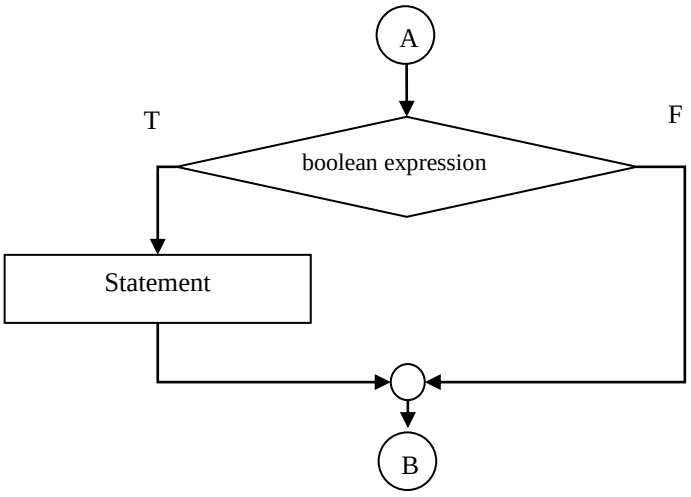
```
int a, b = 12;
{
        int x , y = 8 ;
        {
                int k=0;
                x = a * y;
                k=k+z; //error
        }
}
{
        int z =12;
        a = z + b;
        {
                int u=0;
                x = a * y; //error
                u++;
        }
}
{
        int u = 1 ;
        a= u - b;
        z = u - b ; //error
}
```

## 4- الأمر الشرطي if

يأخذ الأمر if في لغة C# ثلاثة اشكال:

### الشكل الأول: الشكل وحيد الاختيار single selection

يتيح هذا الشكل اختيار تنفيذ أمر ما أو تخطيه. ويكون المخطط التدفقي الممثل لهذه الحالة




**الشكل 5- الشكل وحيد الاختيار**

**الصيغة العامة لهذا الشكل تكون على النحو:**

***if ( boolean_expression )***
***statement***

حيث أن :

if كلمة مفتاحية.

statement هي عبارة عن أمر برمجي ( قد يكون بسيطاً أو مركباً ).

سيتم تنفيذ الأمر statement في حال كون التعبير المنطقي يأخذ القيمة true وإلا سيتم تجاهله وتخطيه.

**مثال**

يتم قبول الطلاب في مرحلة رياض الأطفال إذا كان عمرهم يقل عن 6 سنوات، يمثل المقطع التالي اختباراً فيما إذا كان الطفل سيقبل في صفوف رياض الأطفال وإخبارنا بذلك برسالة.

```
int age;
age = Int32.Parse(Console.ReadLine());
if (age < 6)
      Console.WriteLine("ACCEPTED");
```

قبل المتابعة نود التنويه إلى أن الأمرين التاليين:

**أولاً**– قد يكون التعبير المنطقي بسيطاً ( كما في حالتنا هذه )، وقد يكون مركباً، كأن نقول مثلاً أن قبول الطلاب يكون لمن كانت أعمارهم بين 3 و 5 سنوات، عندها يمكن تعديل التعبير المنطقي في بحيث يصبح:

```
((age < 6) && (age > 2))
```

أو بشكل آخر:

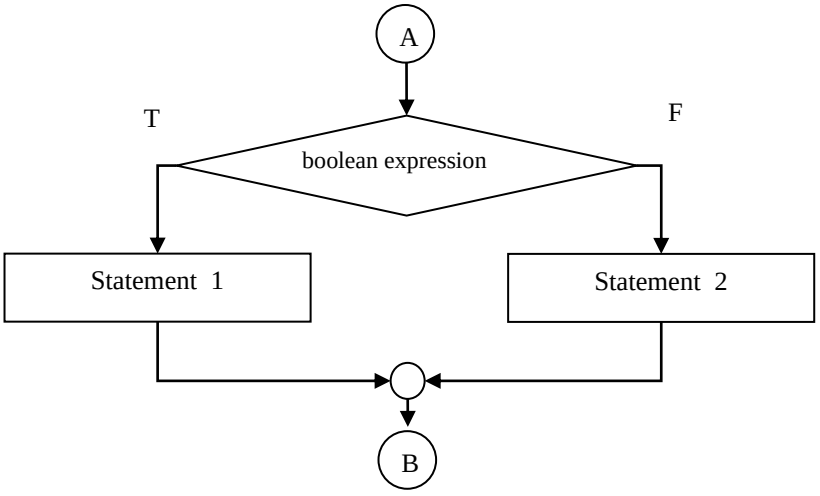
```
((age <= 5) && (age >= 3))
```

**ثانياً**– قد يكون الأمر المراد تنفيذه في حال تحقق الشرط بسيطاً، وقد يكون مركباً، كأن نقول مثلاً أن المطلوب إخبارنا بقبول الطالب وتبيان عدد الطلاب الذين قبلو حتى الآن، عندها لابد من تعديل البرنامج بحيث يصبح:

```
int age;                    //student age
int std_count=0;      // count of accepted
age = Int32.Parse(Console.ReadLine());
if ((age < 6) && (age>2))
{
       Console.WriteLine("ACCEPTED");
      std_count++;
}
Console.WriteLine("{0} \t was accepted",std_count);
```

## الشكل الثاني: الشكل مضاعف الاختيار double selection

يتيح هذه الشكل تنفيذ أحد أمرين وتخطي الآخر. ويكون المخطط التدفقي في هذه الحالة




**الشكل 6– الشكل مضاعف (ثنائي) الاختيار**

**الصيغة العامة لهذا الشكل تكون على النحو:**

```
if ( boolean_expression )
        statement 1
else
        statement 2
```

حيث أن:

if ، else كلمات مفتاحية.

statement هي عبارة عن أمر برمجي ( قد يكون بسيطاً أو مركباً ).

سيتم تنفيذ الأمر statement 1 في حال كون التعبير المنطقي يأخذ القيمة true وإلا سيتم سيتم تنفيذ الأمر statement 2.

**مثال**

بالعودة إلى المثال السابق، قد يكون مطلوباً معرفة فيما إذا قبل الطالب أم لا مع عدد الذين تم قبولهم وعدد الذين لم يتم قبولهم

في هذه الحالة، لابد من استخدام الشكل مضاعف الاختيار كما يلي:

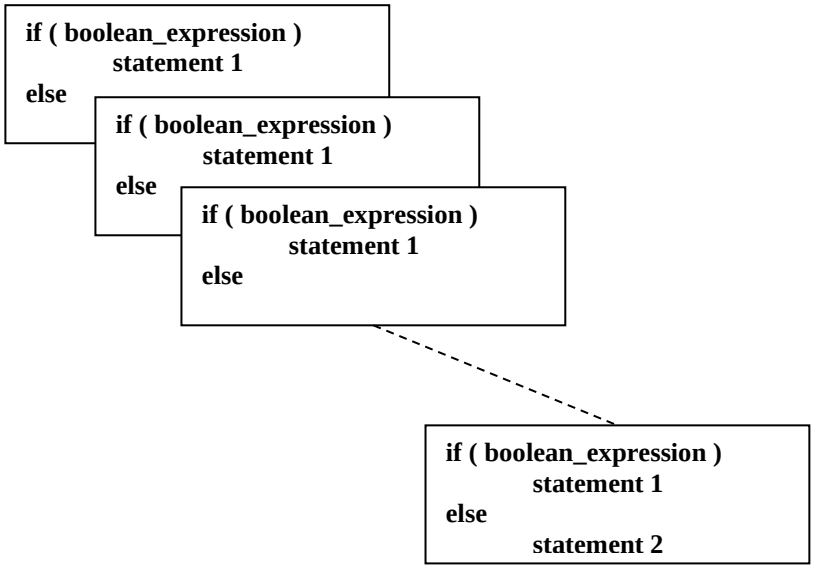
```
int age;                    //student age
int std_count=0;      // count of accepted
int nstd_count=0;      // count of accepted
age = Int32.Parse(Console.ReadLine());
if ((age < 6) && (age>2))
{
       Console.WriteLine("ACCEPTED");
      std_count++;
}
else
{
       Console.WriteLine("REFUSED");
      nstd_count++;
}
Console.WriteLine("{0} \t was accepted",std_count);
Console.WriteLine("{0} \t was accepted",nstd_count);
```

## الشكل الثالث: الشكل متعدد الاختيار multiple selection

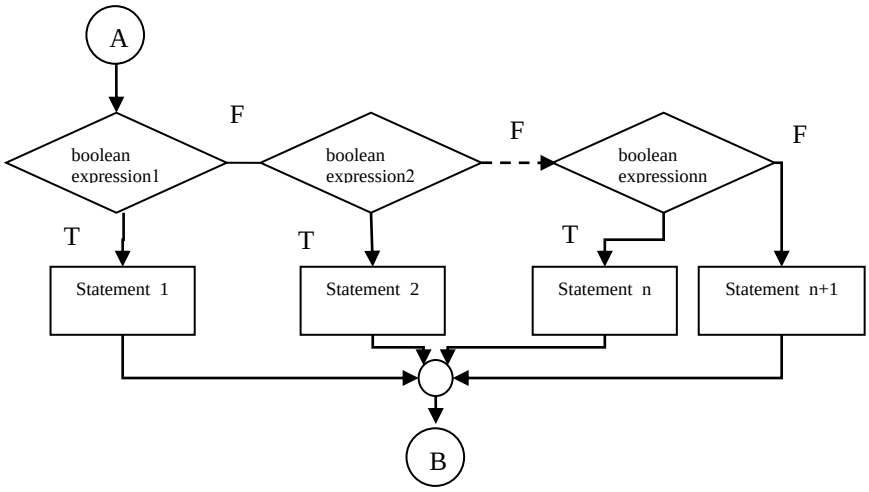
أشرنا سابقاً إلى أن statement ضمن الأشكال السابق للأمر if من الممكن أن يكون أمراً بسيطاً أو مركباً، وفي حالة خاصة، يمكن أن يكون هذا الأمر هو عبارة عن أمر الاختيار if من الشكل مضاعف الاختيار.... وهكذا.

وبالتالي فإن الشكل متعدد الاختيار هو عبارة عن سلسلة من تعليمات if المتداخلة nested المكتوبة كتعليمة واحدة، لذلك يدعى هذا الشكل أحياناً بالشكل المتداخل nested selection.

يمكن توضيح هذا الشكل وفق المخطط التالي:




**الشكل 7- الشكل المتداخل للأمر if**




**الشكل 8- الشكل متعددالاختيار**

**الصيغة العامة لهذا الشكل تكون على النحو:**

```
if ( boolean_expression 1)
        statement 1
else if ( boolean expression 2)
        statement 2
.
.
.
else if ( boolean expression n)
        statement n
else
        statement n+1
```

حيث أن:

if و else هي كلمات مفتاحية.

statement 1، statement 2، . . .هي أوامر بسيطة أو مركبة.

في الشكل الثالث، إذا كان boolean_expression 1 صحيحاً، ينفذ الأمر statement 1 وتجاوز كل ما تبقى، وإلا إذا كان boolean expression 2 صحيحاً، يتم تنفيذ statement 2 وتجاوز بقية الأوامر ... وهكذا.

**مثال**

بالعودة إلى مثال أطفال الروضة، ففي الحقيقة، فإن طلاب الروضة لا يوضعون جميعاً في نفس الصف، وإنما يوضع الأطفال الذين يبلغ عمرهم 3 سنوات في الصف "KG1"، والـ 4 سنوات في الصف "KG2"، الـ 5 سنوات في صف "KG3"،

وبالتالي بتعديل البرنامج بحيث يراعي هذا الأمر نجد:

```
int age;                    //student age
int nstd_count=0, kg1_count=0, kg2_count=0, kg3_count=0;
age = Int32.Parse(Console.ReadLine());
if ((age >=3) && (age<4))
        Console.WriteLine(++kg1_count + "ACCEPTED IN KG1");
else if ((age >=4) && (age<5))
        Console.WriteLine(++kg2_count + "ACCEPTED IN KG2");
else if ((age >=5) && (age<6))
        Console.WriteLine(++kg3_count + "ACCEPTED IN KG3");
else
        Console.WriteLine(++nstd_count + "REFUSED");
```

## عموض التعليمة الشرطية (مشلكة else المعلقة):

رأينا للتو أن الشكل متعدد التفرع للتعليمة if هو في الحقيقة شكل if-else :

```
if ( boolean_expression )
        statement1
else
        statement2
```

حيث statement2 هي تعليمة if أخرى. ولكن ماذا لو كانت statement1 هي تعليمة if أخرى، على سبيل المثال،

```
if ( x > 0 )
    if ( y > 0 )
        z = x + y;
```

وعندما تتبع مثل هذه التعليمة if المتداخلة بـ else، فإن تحديد أي if ترتبط بهذه الـ else ليس أمراً بديهياً.

```
if ( x > 0 )
    if ( y > 0 )
        z = x + y;
else
    Console.WriteLine("\n*** Unable to compute z!");
```

أو

```
if ( x > 0 )
    if ( y > 0 )
        z = x + y;
    else
    Console.WriteLine("\n*** Unable to compute z!");
```

إن هذا الالتباس هو ما يدعى باسم مشكلة الـ else المعلقة، وقد قامت لغة C# بحلها من خلال القاعدة التالية:

**في تعليمة if متداخلة، فإن else ترتبط بأقرب if سابقة غير مرتبطة.**

وبالتالي، ففي تعليمة if السابقة، الصيغة الثانية هي الصحيحة. أي أن else مرتبطة بتعليمة if الداخلية. لو رغبنا بربط else بتعليمة if الخارجية، يمكن القيام بذلك من خلال قسر هذا الربط بوضع تعليمة if الداخلية ضمن أقواس كما يلي:

```
if ( x > 0 )
{
    if ( y > 0 )
        z = x + y;
}
else
    Console.WriteLine("\n*** Unable to compute z!");
```

## 5- أمر الاختيار المتعدد switch

رأينا في الفقرة السابقة أن بالإمكان استخدام الشكل الثالث للأمر if من أجل تحقيق عملية الاختيار المتعدد، حيث يتم إجراء الاختيار من خلال تقييم تعبير منطقي واحد أو أكثر. وبما أن شروط الاختيار يمكن أن تصاغ كتعابير منطقية، يمكن استخدام الشكل if-else-if لتحقيق أي عملية اختيار متعدد.

تستعمل لغة C# أمراً آخر لتحقيق الاختيار المتعدد وهو الأمر switch. وعلى الرغم من أنه لا يملك العمومية التي يملكها الأمر if، إلا أنه أكثر فعالية في تحقيق أشكال معينة من الاختيار.

**الصيغة العامة للأمر switch تكون على النحو:**

```
switch ( expression )
{
   case_list_1 :
          statement_list_1; break ;
   case_list_2 :
          statement_list_2; break ;
    .
    .
   .
   case_list_n :
          statement_list_n; break ;
   default :
          statement_list_n+1; break ;
}
```

حيث أن:

switch و default و break كلمات مفتاحية.

expression هو التعبير أو المتحول المراد اختباره.

كل case_list_i هي تتالي من الحالات من الشكل : case constant_value

العبارة default هي اختيارية.

كل statement_list_i هي تتالي من الأوامر.

يقوم الأمر switch بتقييم expression. إذا كانت قيمة expression من ضمن الـ case_list_i، يبدأ تنفيذ statement_list_i ويستمر لحين الوصول إلى الأمر break. أما إذا كانت قيمة expression ليست من ضمن الـ case_list_i، عندئذ يتم تنفيذ statement_list_n+1 المحددة في العبارة default.

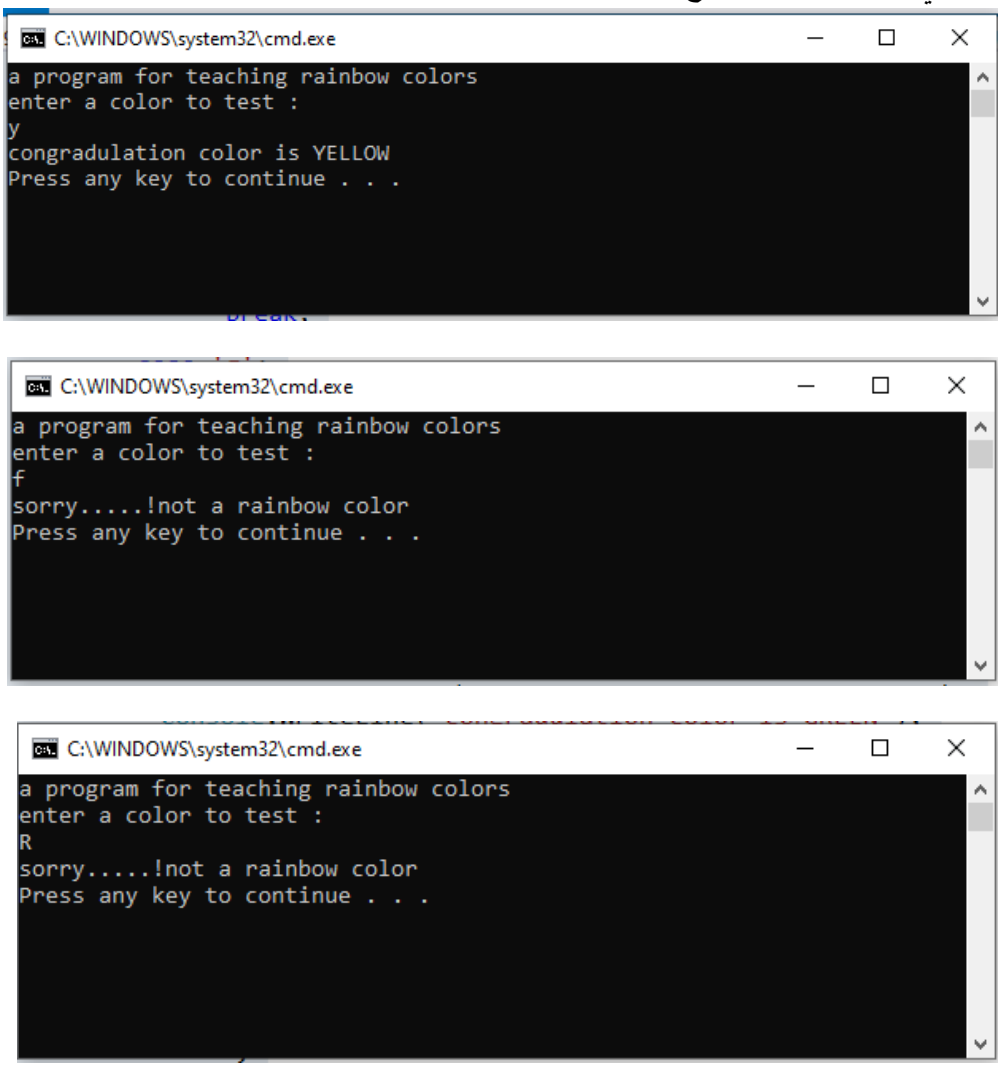
- إذا تم حذف العبارة default ولم تكن قيمة expression ضمن أي case_list_i فإن التنفيذ سيتجاوز الأمر switch بدون فعل أي شئ.
- يؤدي الأمر break إلى الخروج من الأمر switch والانتقال إلى الأمر التالي ضمن البرنامج.
- إن الأمر switch أقل عمومية –كما أسلفنا– من الشكل متعدد الاختيار للأمر if وذلك نظراً لكون نمط الاختبار الوحيد المتاح في هذا الأمر هو اختبار المساواة equality، في حين مع الأمر if فكل أنماط الاختبارات متاحة.

**مثال:**

نرغب بكتابة برنامج يقوم بتعليم مستخدمه ألوان قوس قزح، حيث يتطلب هذا البرنامج من مستخدمه إدخال الحرف الأول من اللون، فإن المحرف المدخل يمثل الحرف الأول في أحد ألوان قوس قزح يقوم البرنامج بإرسال تهنئة تتضمن اسم اللون، وإلا يقوم بإرسال رسالة تفيد بخلاف ذلك.

```
Console.WriteLine("a program for teaching rainbow colors");
char color;
Console.WriteLine("enter a color to test :");
color = Char.Parse(Console.ReadLine());
switch (color)
{
        case 'r':
          Console.WriteLine("congradulation color is RED");
          break;
        case 'b':
          Console.WriteLine("congradulation color is BLUE");
          break;
        case 'g':
          Console.WriteLine("congradulation color is GREEN");
          break;
        case 'y':
          Console.WriteLine("congradulation color is YELLOW");
          break;
        case 'o':
          Console.WriteLine("congradulation color is ORANGE");
          break;
        case 'v':
          Console.WriteLine("congradulation color is VIOLET");
          break;
        case 'w':
          Console.WriteLine("congradulation color is BROWN");
          break;
        default:
          Console.WriteLine("sorry.....!not a rainbow color");
          break;
    }
```

فيما يلي عينة من تنفيذ البرنامج لثلاث مرات:




**الشكل 9- عينة من تنفيذ البرنامج**

بالنظر إلى ناتج التنفيذ، نلاحظ أن التجربة الثالثة تقودنا إلى نتيجة غير سليمة، بالرغم من أن هناك لوناً يبدأ بالمحرف R، إلا أننا نعيد التذكر بأن لغة C# تميز بين المحارف الكبيرة والصغيرة.

يمكن تجاوز هذا الأمر من خلال جعل البرنامج يستجيب للمحرف المدخل سواء كان كبيراً أو صغيراً، وذلك عن طريق توسيع الـ case_list على النحو التالي:

```
……..
switch (color)
   {
      case 'r':case 'R'
               Console.WriteLine("congradulation color is RED");
           break;
……..
```

في هذه الحالة، يكون سلوك الأمر switch مكافئاً لما يلي:

```
if ((color==’r) || (color==’R’))
                Console.WriteLine("congradulation color is RED");
```

وعلى العموم، يمكن لـ case_list أن تحتوي أي عدد من القيم حيث الشكل العام:

```
case ‘value1’ : case ‘value2’: …… case ‘valueN’:
```

**ملاحظة:** كان من الممكن حل المسألة السابقة باستخدام الأمر if -else-if كما يلي:

```
Console.WriteLine("a program for teaching rainbow colors");
char color;
Console.WriteLine("enter a color to test :");
color = Char.Parse(Console.ReadLine());
if (color=='r' || color=='R')
                Console.WriteLine("congradulation color is RED");
else if  (color=='b' || color=='B')
                Console.WriteLine("congradulation color is BLUE");
else if  (color=='g' || color=='G')
                Console.WriteLine("congradulation color is GREEN");
else if  (color==y' || color=='Y')
                Console.WriteLine("congradulation color is YELLOW");
else if  (color==o' || color=='O')
                Console.WriteLine("congradulation color is ORANGE");
else if  (color=='v' || color=='V')
                Console.WriteLine("congradulation color is VIOLET");
else if  (color=='w' || color=='W')
                Console.WriteLine("congradulation color is BROWN");
else
                Console.WriteLine("sorry.....!not a rainbow color");
```

## 6- المعامل الثلاثي Trenary Operator

**المعامل الثلاثي Trenary Operator** هو تعبير ينتج أحد قيمتين، وذلك بحسب قيمة تعبير منطقي (يدعى أيضاً شرط condition ).

**يأخذ المعامل الثلاثي الصيغة العامة التالية:**

***condition ? expression1 : expression2***

حيث:

- condition هو تعبير منطقي.
- expression1 و expression2 هما تعبيران متوافقان بالنوع.

يتم تقييم condition، إذا كانت قيمته هي true فإن قيمة expression1 تتم إعادتها كنتيجة وإلا تتم إعادة قيمة expression2.

وهي صيغة مكافئة تماماً للأمر :

***if (condition ) expression1 else expression2***

لتوضيح ذلك، نعود إلى مسألة قبول الطلاب في مرحلة الروضة، ولنحاول تجربة المقطع التالي:

```
int age;                    //student age
int std_count=0,nstd_count = 0;
age = Int32.Parse(Console.ReadLine());

//first example
Console.WriteLine(age < 6 ? "ACCEPTED" : "REFUSED");

//second example
Console.WriteLine(age < 6 ? ++std_count+"\tACCEPTED" : ++nstd_count+"\tREFUSED");

//third example
string result = age < 6 ? "ACCEPTED" : "REFUSED";
Console.WriteLine(result);
```

## 7- تمارين وأنشطة

**تمرين 1-** ماذا ستكون قيمة m بعد تنفيذ كل من التعليمات التالية

```
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;
```

**تمرين 2-** ماذا هو ناتج تنفيذ البرامج التالية:

```
int a=100, b=0, c=8;
if ( b == 0 )
        b+=3;
if ( a <= 0)
        a= a + b;
else
        a = a * b;
Console.WriteLine("a = " + a);
```

```
int a=0,b=0,c=0 ;

a =   a == 0  ?  b  :  a+1 ;
b =   a >= 0  ?  b-1  :  b+1 ;
c =   (a+b)>0 ? a++ : a--;

Console.WriteLine("a = " + a);
Console.WriteLine("b = " + b);
Console.WriteLine("c = " + c);
```

**تمرين 3-** أكتب برنامجاً لإدخال عددين من لوحة المفاتيح واختبار فيما إذا كان أحدهما من مضاعفات الثاني.

**تمرين 4-** نريد كتابة برنامج لحلّ للمعادلة من الشكل $a.x^2+b.x+c=0$ .

**ملاحظة:** يطلب نقاش حالة (a=0).

**تمرين 5**– أكتب برنامج باستخدام الأمر switch لإدخال رقم الشهر في السنة وطباعة إسمه الكامل.

**تمرين 6**– أكتب برنامجاً يقوم بقراءة محرف من لوحة المفاتيح ثم يقوم بطباعة العبارة:

This is a vowel

إذا كان المحرف هو أحد الأحرف الصوتية ( أي a,e,i,o,u ). وإلا يطبع العبارة:

This is an operator

إذا كان هذا المحرف هو أحد المعاملات الحسابية ( أي %,/,*,-,+ ). وإلا يطبع العبارة:

This is a letter

**تمرين 7**– أكتب برنامجاً يقوم بقراءة محرف وعددين صحيحين، حيث يعبر المحرف عن معامل حسابي ما، والعددين الصحيحين هما القيمتان اللتان سيتم تطبيق العملية الحسابية عليهما.

يجب أن يقوم البرنامج بإعادة الناتج.