Capitolo 5
Controllo di flusso

Quando si programma con un linguaggio di alto livello si ha prima o poi a che fare con istruzioni che aiutano a compiere delle scelte o a effettuare attività di controllo o ripetitive. C# non fa eccezione e ci propone una serie di istruzioni utili allo scopo. Chi conosce già altri linguaggi non troverà probabilmente nulla di particolare.

Comiciamo a presentare i comandi decisionali, simili semanticamente tra di loro, che sono **if** e **switch**.

if
Questa istruzione basa la sua esecuzione sul valore di una espressione booleana e, si noti, solo booleana, non sono ammessi valori tipo 0 e 1 come in C/C++. Il costrutto si presenta nel suo formato generale come segue:

caso 1

if *(condizione)*
istruzioni
else
istruzioni

oppure, caso 2

if*(condizione)*
istruzioni
else if *(condizione)*
istruzioni
else if *(condizione)*
istruzioni
....
else
istruzioni.

Nel primo caso abbiamo sistuazione a due vie: la prima si realizza se la condizione risulta true altrimenti si entra nel ramo alternativo introdotto da **else**. Nel secondo caso abbiamo varie ramificazioni che vengono eseguite a seconda di quale è la condizione che si rivela true, altrimenti si termina nell'else finale. Le parole chiave che entrano in gioco pertanto sono solo due **if** ed **else** oltre alla loro combinazione **else if**. Le istruzioni ovviamente devono essere comprese all'interno di una coppia di parentesi graffe se sono di numero maggiore di uno, si tratta di un normale blocco come quelli già incontrati, in questo senso il fatto di essere all'intero di un costrutto come if non cambia nulla. Vediamo un semplice esempio:

C# Esempio 5.1

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      Console.Write("Inserisci la tua media voti: ");
7      int x = Int32.Parse(Console.ReadLine());
8      if (x <= 0)
9      Console.WriteLine("Numero troppo basso");
10     else if (x > 0 && x <= 10)
11     Console.WriteLine("Siamo nella media");
12     else
14     {
15       Console.WriteLine("Numero troppo alto");
16       Console.WriteLine("Bugiardone!!");
17     }
```

```
18      }
19  }
```

L'esempio è abbastanza banale (anzi molto banale, lo riconosco) e si riferisce al secondo caso di configurazione di un if. La riga 8 ci presenta la prima condizione booleana; se questa non si realizza il programma testa la condizione alla riga 10 e se neanche questa è verificata viene eseguito l'else alla riga 12 che ha sotto di se un codice costituito da due righe. Come si può vedere si tratta di una istruzione molto semplice sia concettualmente che in pratica. Va notato come il realizzarsi di una condizione esclude le altre successive che vengono ignorate anche qualora fossero realizzate (in tal caso ovviamente il codice sarebbe da rivedere un po').

Una osservazione che è bene fare è che la type safety di C# ci protegge da uno degli errori più comuni, come ben sa che arriva da C/C++ ovvero dimenticare un "=" e scrivere cose tipo:

if (x = 5)....

invece di if (x == 5) che è la versione corretta. Nel primo caso il compilatore si farà sentire. Un singolo segno =, come già ricordato nel capitolo 4 equivale ad un assegnamento mentre due segni = costituiscono l'operatore di confronto per l'uguaglianza. Sembra una piccola cosa ma in realtà è veramente molto utile in termini di sicurezza contro noiosi bug.

E' possibile nidificare uno o più if all'interno dell'altro

C# Esempio 5.2

```
1   using System;
2   class program
3   {
4     public static void Main()
5     {
6       Console.Write("Inserisci un numero: ");
7       int x = Int32.Parse(Console.ReadLine());
8       Console.Write("Inserisci un altro numero: ");
9       int y = Int32.Parse(Console.ReadLine());
10      if (x > 10)
11        if (y > 10) Console.WriteLine("Entrambi maggiori di 10");
12        else Console.WriteLine("Solo il primo maggiore di 10");
13      else
14        if (y > 10) Console.WriteLine("Solo il secondo maggiore di 10");
15        else Console.WriteLine("Entrambi minori o uguali a 10");
16    }
17  }
```

Qui c'è un certo intreccio di if ed else. Quindi quello alla riga 12 funge da alternativa all'if che trova alla 11 mentre quello alla riga 13 si riferisce all'if alla 10 mentre quello alla riga 15 ha il suo if corispondente alla riga 14. Indentando il codice questo diventa più leggibile, altro esempio del perchè è bene ricorrere all'indentazione nei propri programmi. Comunque attenzione in casi come questi ad una attenta valutazione delle casistiche dei vari rami.

Nel capitolo dedicato agli operatori attivi in C# vedremo un sistema di scrittura alternativo basato sull'operatore **?**

Non c'è altro da dire sul nostro if. Tenete presente comunque che questa istruzione rende comunque sempre un po' pesante il codice e a volte di difficile manutenzione specie se avete molte ramificazioni dal momento che comunque non vi sono limiti al numero di if-else che potete codificare. In tal caso bisognerebbe considerare il possibile uso di altri costrutti o rivedere l'architettura del programma in quel punto.

switch

Compagno di merende ideale per if è lo statement **switch** che effettua esso pure una selezione basandosi su un certo insieme di scelte. In particolare questo insieme deve essere costituito dai valori assunti da un variabile di tipo **intero** o **stringa.** Le parole chiave interessate sono 4: **switch**, **case**, **break**, **default** mentre la struttura generica del costrutto è la seguente:

***switch** (selezionatore)*
***case** caso1:*
istruzioni
***break**;*
***case** caso2:*
istruzioni
***break**;*
.....
***default**:*
istruzioni
break

Il selezionatore è la variabile che, come detto, sarà di tipo **intero** o **stringa** mentre le varie clausole **case** riportano ai valori assunti dalla stessa ovvero caso1, caso2 ecc... questi rappresentano specificamente il valore della variabile che innesca un certo ramo della selezione; quindi a seconda dei valori assunti dal selezionatore si passerà il controllo al ramo corrispondente. Diversamente, se nessun caso specificato dalla cluausola "case" si verifica allora viene realizzato il ramo **default**. Importante è al presenza della istruzione break il cui fine è uscire dallo switch senza interprellare le altre possibilità ovvero altri rami oltre a quello che realizza scelta. Si tratta di una sorta di misura di sicurezza per evitare il fenomeno cosiddetto "**fall-through**" che si verifica in altri linguaggi, come C++ e anche Java, dove è possibile eseguire due sezioni diverse di uno switch; questa è funzionalità potente ma anche fonte di bugs piuttosto difficili e noiosi da rilevare in caso di dimenticanze. L'assenza di break in C# viene segnalata come errore da parte del compilatore. Il fall-through, se proprio necessario, può essere emulato come vedremo poco più avanti L'istruzione switch è forse più elegante e gestibile di if che da parte sua è più flessibile.

Vediamo il solito esempio di base:

C# Esempio 5.3

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      Console.Write("Inserisci un numero: ");
7      int x = int.Parse(Console.ReadLine());
8      switch(x)
9      {
10       case 1:
11       Console.WriteLine("Hai inserito in numero 1");
12       Console.WriteLine("bassissimo");
13       break;
14       case 2:
15       Console.WriteLine("Hai inserito il numero 2");
16       break;
17       default:
18       Console.WriteLine("Numero maggiore di 2");
19       break;
20     }
21   }
22 }
```

Switch non ha molte particolarità ma un paio esistono ed una è curiosa. La prima variante è che se vogliamo che più risultati abbiano lo stesso esito è necessario procedere a una modifica struttura come nell'esempio che segue

```
Console.Write("Inserisci un numero: ");
int x = int.Parse(Console.ReadLine());
switch(x)
{
  case 1:
  case 2:
  case 3:
  case 4:
```

```
  Console.WriteLine("Hai inserito un numero minore di 5");
  break;
  default:
  Console.WriteLine("Numero maggiore o uguale a 5");
  break;
}
```

Forse, almeno a parer mio, non è proprio il massimo dell'eleganza ma questo passa il convento... Come seconda particolarità va segnalato come sia permesso il salto da un ramo all'altro, quindi se necessario anche emulando un fall-through, tramite il bistrattatissimo operatore **goto**, ovvero il comando più esecrato nel mondo della programmazione, direi anche in maniera eccessiva. Se volete farvi cacciare da qualunque colloquio finalizzato ad assunzione o collaborazione con qualsiasi software house provate a tessere qualche elogio del goto. **Tanenbaum** e **Jacopini-Bohem** sono alcune delle voci che danno forza al dilagante movimento anti-goto. Curiosamente però quasi tutti i compilatori se la tengono stretta come parola riservata... non si sa mai. Detto questo tuttavia in un caso come quello seguente il suo uso ha un suo perchè. L'istruzione goto di per se stessa è presto spiegata: essa rimanda attraverso un salto ad una etichetta, con nome a scelta dell'utente seguita da due punti

label:
istruzioni
.....
goto label;

e il programma salta al blocco di codice che segue immediatamente label. Aggiungo che il goto può essere usato per saltare fuori da un livello più interno del codice (ma non fuori da una classe o da un blocco che inizi con la keyword finally) ma non dall'esterno verso l'interno. L'esempio che segue completa il discorso sullo switch e illustra contestualmente il goto:

C# Esempio 5.4

```
1  using System;
2
3  class test
4  {
5      public static void Main()
6      {
7          Console.Write("Inserisci un numero: ");
8          string s = Console.ReadLine();
9          int x = int.Parse(s);
10         switch(x)
11         {
12             case 1: label1: Console.WriteLine("Il valore è > 0"); break;
                   // abbiamo inserito la label1
13             case 2: Console.WriteLine("il valore è 2");
14             goto label1; break;
                   // il compilatore segnala un warnig qui si può ignorare
15             default : Console.WriteLine("Valore > 2"); break;
16         }
17     }
18 }
```

Il goto compare alla riga 14 e, nel caso in cui si sia inserito il valore due oltre a stampare la relativa scritta rimanda anche al caso dell'etichetta 1. Il goto è l'unico modo per ecuperare quella porzione di codice. Questo non è comunque un buon esempio di programmazione.

L'istruzione switch non aggiunge nulla da un punto di vista semantico ed espressivo al linguaggio rispetto all'if e può essere sostituita da questo. Tuttavia switch in molti casi è più leggibile e anche più efficiente in quanto riduce i confronti da effettuare.

Concludiamo, per completezza di informazione il paragrafo dedicato a switch con un esempio che si basa su una selezione compiuta si stringhe invece che su numeri, la logica ovviamente non cambia:

C# Esempio 5.5

```
1  using System;
2
3  class test
4  {
5      public static void Main()
6      {
7          Console.Write(Nnome della tua squadra di calcio preferita?: ");
8          string s = Console.ReadLine();
9          switch(s)
10         {
11             case "Juventus": Console.WriteLine("Buona scelta!); break;
12             case "Milan": Console.WriteLine("Ma noooo"); break;
13             case "Inter": Console.WriteLine("Peggio di così...");break;
14             default : Console.WriteLine("Chi vive sperando..."); break;
15         }
16     }
17 }
```

E' il momento di analizzare gli operatori di iterazione che in C# sono 4 (non vorrete fare un loop con label e goto vero?) e precisamente **for**, **while**, **do while** e **foreach**.

FOR

Il ciclo for è un altro comando assai "famoso" nel mondo della programmazione. Praticamente in tutti i linguaggi esiste questa istruzione che esprime in modo molto semplice la ripetizione di comandi per un numero di volte stabilito dal programmatore, al limite anche infinito. Molto spesso for è usato per effettuare cicli su array e altri costrutti indicizzati. La condizione di uscita è stabilita dal raggiungimento del valore di false in una data espressione booleana. Il costrutto generale è il seguente:

***for** (condizione)*
istruzioni

dove condizione è normalmente nella forma:
variabile; espressione booleana; incremento/decremento.
L'esempio qui di seguito serve a chiarire questo concetto:

C# Esempio 5.6

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      Console.Write("Inserisci un numero: ");
7      int x = Int32.Parse(Console.ReadLine());
8      for (int i = 0; i <= x; i++)
9      {
10       Console.WriteLine("Passo # {0}",i);
11       Console.WriteLine("Mancano {0}", (x - i) + " passi alla fine");
12     }
13   }
14 }
```

La riga 8 è quella che ci interessa ed introduce un uso comunissimo del ciclo for. La condizione tra parentesi costituisce la parte più interessante e segue lo schema precedentemente definito. Viene dichiarata una nuova variabile di tipo intero che, subito dopo, è confrontata con il valore x ricavato in precedenza. Ecco introdotta la condizione di terminazione quel i <= x che in pratica equivale qui ad un if i <= x. La variabile i viene poi incrementata altrimenti la condizione di uscita non viene raggiunta. Tutto molto semplice. Interessante è anche vedere che fine fa la variabile i. Se infatti prima della chiusura del main aggiungessimo, quindi diciamo tra la 12 e la 13, fuori dal loop, la seguente riga:

```
Console.WriteLine(i);
```

il compilatore ci risponderebbe che non vede la variabile i. Questo ovviamente fa parte delle regole di "scope", di visibilità delle variabili. i è creata all'interno di for e non è accessibile altrimenti, questo in piena sintonia con quanto detto al capitolo 3 relativamente allo scope delle variabili. In conformità con quel concetto è evidente che il ciclo for è un blocco chiuso e pertanto una variabile definita al suo interno non può essere vista all'esterno, ad un livello più alto da qui la indisponibilità della variabile i.

L'incremento può essere anche maggiore di uno. Ad esempio la riga 8 poteva essere scritta:

```
for (int i = 0; i <= x; i+=2)
```

che incrementa di due unità per volta il valore di i. Inoltre è anche possibile lavorare su più di una variabile per volta come nel seguente esempio:

C# Esempio 5.7

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      for (int x = 0, y = 0; x <= 10 && y <= 10; x++, y++)
7      Console.WriteLine("x = " + x + " y = " + y + " -> " + " x + y = " + (x + y));
8    }
9  }
```

dove la condizione di confronto deve essere vera sia per x che per y come richiesto dall'operatore && (AND). Ovviamente in questi casi bisogna fare un po' più di attenzione ai test che la condizione di confronto deve soddisfare.

L'istruzione for si presta molto bene ad essere innestata in altri cicli for, ottimo per la gestione di base di matrici, per fare un esempio pratico:

```
for (i = 0; i < 10; i++)
for (x = 0; x < 10; x++)
istruzioni relative al for innestato;
ecc....
```

Ovviamente, banale ma vale la pena ricordarlo, come da definizione fornita in fase introduttiva, possiamo avere un decremento invece di un incremento:

```
for (x = 10; x > 1; x--)
```

Può a volte essere utile utilizzare il for per realizzare un loop infinito. La cosa è molto semplice, anche senza ricorrere ad astrusità in termini di codice:

```
for(;;)
istruzioni
```

in caso caso il gruppo di istruzioni viene ripetuto all'infinito laddove non contenga un qualche costrutto di interruzione, ad esempio il break che vedremo più avanti. Così è più chiaro e leggibile che non scrivere cose (come ho visto) tipo

```
for (int x = 5; x < 3; x++)...
```
☺

Come ultima considerazione poniamo attenzione al fatto che l'incremento (e, del tutto analogamento il decremento), può essere scritto ++x (--x) anzichè x++ (x--) posto che x sia la variabile indice. In questo caso il risultato può essere leggermente diverso, si salta una iterazione in pratica, come è facile verificare costruendosi un esempio.

WHILE

questo comando esegue un blocco di istruzioni fino a che non risulta false una condizione di confronto. La sintassi è:

while *(espressione)*
istruzioni

l'espressione come detto è di tipo booleano e costituisce il confronto per terminare l'esecuzione del ciclo. E' importante notare come il confronto avvenga prima di eseguire il blocco di istruzioni per cui questo potrebbe non venire mai eseguito.

C# Esempio 5.8

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      int x = 10;
7      int i = 1;
8      while (i < x)
9      {
10       Console.WriteLine(i);
11       i++;
12     }
13   }
14 }
```

L'output del programma è dato dai numeri da 1 a 9. la riga 8 introduce la nostra istruzione seguita dalla condizione booleana che viene valutata in ingresso e poi ad ogni ciclo. L'incremento alla riga 11 permette il raggiungimento della condizione di uscita che viene ottenuta quando la variabile i raggiunge il valore 10. Invertendo ad esempio il verso del confronto (i > x) anzichè (i < x) il codice contenuto nel while non sarebbe mai stato eseguito. Anche con il while è possibile effettuare facilmente dei loop infiniti, ad esempio, poco elegantemente, mettendo come condizione di uscite dei confornti irrealizzabili oppure, più semplicemente, scrivendo

```
while(true)
istruzioni
```

interrompibile eventualmente con l'istruzione break che vedremo tra poco.

while non ha grandi particolarità laddove si abbia un po' di attenzione nella gestione della condizione di uscita al fine di evitare più che altro fastidiosi loop infiniti.

DO - WHILE

è simile in tutto e per tutto a while solo che la valutazione della condzione avviene dopo la prima esecuzione del codice che costituisce il corpo del loop che pertanto si è certi verrà eseguito almeno una volta. Questa è la differenza fondamentale ruspetto al ciclo while. La sintassi è:

do
istruzioni
while *(espressione)*

con quella espressione che troviamo tra parentesi che deve essere, come di consuet,o di tipo booleano.

Vediamo l'esempio relativo al while adattato al do-while

C# Esempio 5.9

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      int x = 10;
7      int i = 1;
8      do
9      {
10       Console.WriteLine(i);
11       i++;
12     }
13     while (i < x);
14   }
15 }
```

Il risultato è identico però stavolta, invertendo la condizione di uscita (ovvero ponendo alla riga 13 (i > x) al posto di (i < x) avremmo comunque una (e una sola, come intuibile) esecuzione del ciclo. Anche in questo caso le uniche attenzioni vanno rivolte ad una corretta gestione della espressione booleana. Indentico al caso del while è il costrutto per realizzare il loop infinito.

FOREACH

Questo comando è un po' più diverso dal punto di vista applicativo anche perchè coinvolge concetti che spiegheremo un poco più avanti. In pratica si tratta di un processo di iterazione in sola lettura, quindi non è possibile modificare o cancellare dati, su array e collections. E' pertanto opportuno rimandare una analisi di questo comando ai capitoli che saranno dedicati agli array prima e alle collections più avanti. Concettualmente comunque non offre particolari difficoltà. La sintassi è la seguente:

foreach *(tipo identificatore* ***in*** *espressione)*
blocco istruzioni

esempio:

```
foreach (int x in arraydiInteri)
{
..........
}
```

Come detto ne riparleremo.

Le prossime istruzioni che presentiamo sono invece dedicate alla interruzione del normale svolgimento di un ciclo. Esse sono **break**, **continue** e **return**. Ci sarebbe anche il **goto** ma ne abbiamo già parlato.

BREAK

Lo abbiamo incontrato parlando dell'istruzione switch. In quel caso serviva a terminare il flusso di esecuzione del comando saltandone fuori. Questo è proprio il suo compito preciso cioè fermare un certo flusso di esecuzione. Normalmente esso termine il loop che lo include passando il controllo all'istruzione successiva al loop. Può essere utilizzato con uno qualsiasi dei comandi visti in precedenza. Nell'esempio che segue lo vediamo all'opera per impedire che un ciclo continui all'infiinito, vediamo come partendo dal codice di base che modificheremo introducendo il break:

C# Esempio 5.10

```
1  using System;
2  class program
3  {
4    public static void Main()
5    {
6      long x = 0;
```

```
7       for (;;)
8       {
9         Console.WriteLine(x);
10        x++;
11      }
12    }
13  }
```

Questo programma non vi darà molte soddisfazioni tranne veder scorrere un interminabile (o quasi, prima o poi arriva un overflow) scia di numeri sullo schermo. Per terminarlo dovrete ricorrere ai classici <ctrl><c> o <ctrl><break>. Il nostro break invece può fare le vostre veci quando lo deciderete: ad esempio modifichiamo il programma come segue:

C# Esempio 5.11

```
1   using System;
2   class program
3   {
4     public static void Main()
5     {
6       long x = 0;
7       for (;;)
8       {
9         Console.WriteLine(x);
10        x++;
11        if (x == 1000) break;
12      }
13    }
14  }
```

Questo breve programma termine spontaneamente quando viene stampato il numero 999 a causa del break.Il controllo passa all'istruzione successiva al loop che in questo caso coincide col termine del programma. Il comando break può essere utilizzato anche all'interno di un while o di un do ... while o anche di un foreach in modo analogo.

CONTINUE

Questa utile istruzione passa il controllo alla iterazione successiva nell'ambito del ciclo che la contiene. Non si tratta quindi di uscire dal loop come permesso da break ma solo, in pratica, di saltarne una o più esecuzioni. Il modo migliore per illustrarne il semplice funzionamento è con un esempio:

C# Esempio 5.12

```
1   using System;
2   class program
3   {
4     public static void Main()
5     {
6       for (int x = 0; x < 10; x++)
7       {
8         if (x == 5) continue;
9         Console.WriteLine(x);
10      }
11    }
12  }
```

l'output è il seguente:

Si nota chiaramente **l'assenza del numero 5** dalla sequenza, questo perchè continue ha istruito il programma a saltare direttamente al passo dell'iterazione successivo a quello corrente, prima quindi che fosse stampato il numero, infatti la scrittura alla riga 9 non viene richiamata.

RETURN

termina l'esecuzione di un metodo restituendo il controllo al chiamante. Qualcosa lo abbiamo già visto nel capitolo 2 ma sarà oggetto di ulteriori approfondimenti quando parleremo dei metodi e relativi valori di ritorno.

Conclusioni.

Le istruzioni qui presentate direi che sono molto semplici da un punto di vista concettuale. Il loro uso è molto ampio e comune tanto è che comandi del tutto simili sono presenti in quasi tutti i linguaggi di programmazione. Non dovrebbe essere difficile fare un po' di pratica con esse, magari partendo dai semplicissimi esempi presentati, sarà senza dubbio un'esperienza utile per future finalità.