INTRODUZIONE

Visual Basic.NET (di seguito VB.NET) è un linguaggio semplice e potente, progettato per unire l'intuitività di Visual Basic con la potenza del .NET. Inizialmente scriveremo codice VB.NET associato a Web Form ma lo stesso potrà essere inserito in applicazioni Windows Form, Classi, Librerie di Codice, ecc.. senza importanti modifiche.
Nel capitolo precedente abbiamo visto come una Web Form inizialmente presenta la pagina del codice con il seguente contenuto:

```
Partial Class _Default
    Inherits System.Web.UI.Page

End Class
```

Quando si scrive un programma, si devono rispettare alcune regole che il linguaggio ci impone. In particolare, ogni programma è strutturato in base a uno "scheletro" predefinito, che nella sua forma più semplice è:

```
Class nome-classe

End Class
```

Nel caso delle Web Form la classe sarà il contenitore di tutto il codice che si scriverà al suo interno e il *nome-classe* identifica in modo univoco la Web Form nel progetto. In particolare, tutto il codice associato a siti anche complessi, costituiti da tante Web Form (pagine), sarà sempre contenuto in contenitori *Class..End Class* ognuno dei quali farà capo alla propria Web Form.

Gli ambienti di sviluppo generano automaticamente lo scheletro del programma nel momento in cui questo viene creato.

Anche se non di particolare rilievo, la seconda riga della dichiarazione della classe:

```
Inherits System.Web.UI.Page
```

indica che la Web Form è un oggetto di tipo *Pagina* e dunque già contenente funzionalità e caratteristiche predefinite. Il progetto, anche se non è stata ancora digitata una riga di codice, è già funzionante; a noi il solo compito di personalizzare la pagina inserendo i controlli desiderati e il relativo codice per la loro gestione.

In molti elementi del linguaggio VB.NET, come per le classi viste precedentemente, si presenta una sintassi simile a quella indicata di seguito:

```
elemento nome
    ciò che sta qui dentro appartiene a nome
End elemento
```

nome viene utilizzato per contrassegnare l'insieme di istruzioni contenute all'interno del "blocco"; Il blocco è chiaramente delineato da un inizio e da una fine. Per una convenzione stilistica chiamata "indentazione del codice sorgente" tutto ciò che sta dentro a un "blocco" viene spostato sulla destra di un numero di spazi fisso.

Un altro blocco molto importante è il blocco di SUB solitamente associato agli eventi dei controlli presenti nel Web Form.

```
sub nome
    ciò che sta qui dentro appartiene a Sub e corrisponde ad una o più istruzioni
End Sub
```

TIPI, VARIABILI ED ESPRESSIONI

Il concetto intuitivo di "tipo di dato" è noto a tutti e fa riferimento alla natura e al significato che diamo all'informazione. Ad esempio, il nome di una persona è un'informazione di natura diversa rispetto alla sua età. Entrambi si scrivono mediante sequenze di simboli (lettere da una parte e cifre dall'altra), ma quelle stesse sequenze assumono due significati diversi.

Le e lettere che compongono un nome non sono soggette ad alcuna interpretazione, semplicemente identificano, mediante la loro combinazione, un certo individuo. Le cifre che compongono un'età esprimono un valore quantitativo preciso e delle relazioni altrettanto precise; ad esempio: un uomo di 25 anni è più giovane di un uomo di 75; 25 è minore di 75.

E ancora, il colore dei capelli di una persona si esprime con una combinazione di lettere ("rossi", "biondi", "scuri", eccetera). E come per i nomi, tali combinazioni non assumono significati particolari al di là della caratteristica (colore dei capelli) che identificano. Nonostante ciò, nessuno compilerebbe mai un elenco fatto per metà di colori di capelli e per metà di nomi di persona; tutti i e due i tipi sono contraddistinti da combinazioni di lettere, ma quelle combinazioni hanno significati diversi tra loro: sono tipi diversi d'informazioni. Infine, un numero di telefono è rappresentato da una combinazione di cifre (perlomeno in Italia) esattamente come un'età, ma assume un significato diverso. Un numero di telefono non designa una quantità di qualcosa; non si confrontano tra loro numeri di telefono, se non per stabilire se sono uguali oppure no.

In conclusione, il concetto che abbiamo di tipo di dato è essenzialmente semantico, si riferisce cioè al significato che assumono le informazioni di quel tipo.

Nei linguaggi di programmazione, e VB.NET non fa eccezione, le cose sono abbastanza diverse. In essi il concetto di tipo di dato si richiama alle modalità di:

1. **memorizzazione e codifica:** come viene codificato in memoria il dato e quanta memoria occupa; qual è, se è definito, l'intervallo di valori che il dato può assumere;
1) **rappresentazione:** in che modo vengono rappresentati i valori costanti, chiamati letterali;
2) **elaborazione:** l'insieme delle operazioni che si possono effettuare con il dato;
2. **relazioni con altri tipi di dati:** compatibilità e conversioni che possono essere effettuate da e verso altri tipi di dati.

VB.NET stabilisce delle precise regole formali che caratterizzano completamente i punti sopra elencati per ogni tipo di dato, al di là del significato che valori di quel tipo possono assumere nella logica del programma.

TIPI DI DATI NUMERICI: INTEGER E DOUBLE

VB.NET mette a disposizione vari tipi di dati numerici; di questi saranno per il momento analizzati il tipo *integer* e *double*. Prima di tutto, però, occorre rispondere ad un quesito: perché avere più tipi numerici? Non sarebbe sufficiente un tipo soltanto? La risposta è: sarebbe sufficiente, ma sarebbe anche inefficiente. Il perché sta nella codifica, nell'occupazione di memoria e nell'efficienza delle operazioni. I numeri vengono inseriti nel codice senza delimitatori, con segno o senza segno e il separatore decimale, ove presente, è il punto. Esempi di valori corretti di costanti numeriche sono 10 11.56 -200.

IL TIPO DOUBLE

Il tipo ***double*** corrisponde all'insieme dei numeri reali e viene definito un tipo a virgola-mobile (floating-point), termine che ne designa la modalità di codifica in memoria. Un valore appartenente al tipo *double* occupa 8 byte di memoria e rientra nel seguente intervallo di variazione (qui espresso attraverso valori positivi):

valore più piccolo: (circa) 5.0×10^{-324}

valore più grande: (circa) 1.7×10^{308}

Ciò significa, ad esempio, che il numero: 1.0×10^{-400} non può essere rappresentato attraverso il tipo double: è un numero troppo vicino allo zero.

Il tipo double possiede una precisione di 16 cifre. Per precisione s'intende il numero di cifre realmente necessarie per distinguere un numero da un altro di grandezza equivalente (cifre significative). Ciò corrisponde al numero di cifre decimali dopo che il numero è stato ricondotto alla forma:

0.<cifre decimali> x 10 <esponente>

Ad esempio:

1.23 contiene 3 cifre significative: 0.123×10^{1}

0.0012 contiene 2 cifre significative: 0.12×10^{-2}

0.1200 contiene 2 cifre significative: 0.12×10^{0}

12340000 contiene 4 cifre significative: 0.1234×10^{8}

Dunque, il seguente numero:

12345678912345678

che contiene 17 cifre diverse da zero viene ricondotto alla forma:

12345678912345670 che corrisponde a: $0.1234567891234567 \times 10^{17}$

forma che non rappresenta l'ultima cifra, e cioè 8.

Rispetto al tipo *integer*, il numero di byte di memoria occupata e il costo computazionale (tempo impiegato dalla CPU per compierla) legato alle operazioni con valori *double* rendono questo tipo appropriato soltanto quando la natura del dato da rappresentare, la sua semantica, è effettivamente di tipo reale. Ad esempio: volumi, altezze, pesi, valori monetari, ma non valori cardinali (numerosità di un insieme) o codici numerici.

IL TIPO INTEGER

Il tipo *integer* corrisponde all'insieme dei numeri interi con segno. Un valore di tipo *integer* occupa 4 byte di memoria e rientra nel seguente intervallo di variazione:

valore più piccolo: –2147483648 -2^{31}

valore più grande: 2147483647 $2^{31}-1$ (tolto lo zero)

Ciò significa, ad esempio, che il numero 10.000.000.000 non può essere rappresentato attraverso il tipo *integer*, poiché è un numero troppo grande. E' appropriato l'uso del tipo *integer* ogni qualvolta la semantica del dato da rappresentare è effettivamente di tipo intero, poiché questo garantisce una minore occupazione di memoria e un minor costo computazionale. Vi sono inoltre alcune operazioni, come la

divisione a quoziente intero e l'accesso con indice a un array, che richiedono espressamente valori di tipo *integer*.

TIPI DI DATI LETTERALI

TIPO STRING

Il tipo *string* consente la rappresentazione d'informazioni alfanumeriche e cioè sequenze di caratteri, che possono rappresentare nomi, indirizzi, codici fiscali, numeri di telefono, eccetera.
Si può immaginare un valore di tipo *string* – detto anche stringa – come una sequenza, o collezione, di caratteri, ognuno dei quali è codificato mediante il codice Unicode (in passato era il codice ASCII) . La memoria occupata da una stringa non è fissa come per i tipi *integer* e *double*, ma dipende dal numero di caratteri, ognuno dei quali occupa due byte.
Ad esempio, la stringa "Salve, Mondo!" è memorizzata nel seguente modo

S	a	l	v	e	,		M	o	n	d	o	!

In realtà, in memoria non vengono codificati i caratteri (cosa impossibile) ma i corrispondenti valori numerici tratti dalla tabella dei codici Unicode.

COSTANTI LETTERALI STRINGA

Una costante letterale stringa viene designata racchiudendo i caratteri che la formano all'interno di una coppia di virgolette. Dunque, letterali stringa sono:

```
"Salve, mondo!"     "ciao come va!"    "Bill Gates"    "055/12345678"
```

E' importante comprendere che è la notazione usata a denotare un letterale stringa e non il valore letterale in sé. Ciò detto, i seguenti letterali:

```
"123"   "0,25"   "10,5E-10"   "-100000"
```

sono costanti stringa e non costanti numeriche. Infatti sarebbe un errore scrivere:

```
RISULTATO: VARIABILE DI TIPO INTEGER

risultato = "10" + 1;       // errore: "10" è una stringa!
```

poiché `"10"` è un letterale stringa e non si può assegnare una stringa ad una variabile di tipo *INTEGER*.

TIPI DI DATI BOOLEANI

Il tipo *boolean*, o **tipo booleano**, è il tipo delle **espressioni condizionali**, chiamate anche **espressioni booleane**, usate nelle condizioni dei Test (costrutto Selezione).
Nel seguente codice:

```
IF (a > b) THEN
        "a è maggiore di b"
ELSE
        "a non è maggiore di b"
END IF
```

la valutazione dell'espressione booleana **a > b** produce un valore di tipo *boolean*, che può essere soltanto **vero** o **falso**, in base al fatto che il valore contenuto in **a** sia o no maggiore del valore contenuto in **b**.
Un valore di tipo booleano occupa un byte di memoria e la sua elaborazione ha un costo computazionale molto basso. Ovviamente, come per tutti i tipi di dati, possono essere dichiarate variabili di tipo *boolean*.

COSTANTI LETTERALI BOOLEANE

VB.NET esprime i due possibili valori che appartengono al tipo *boolean* mediante le costanti: **True** (vero) e **False** (falso).

TIPI DI DATI DATA

Il tipo *date* di VB.NET consente di memorizzare informazioni di tipo data; dichiarando delle variabili di tipo data sono supportate tutte le operazioni algebriche e di confronto su di esse. I valori costanti di tipo data, possono essere assegnati usando la seguente sintassi generale #MM/GG/AAAA#.
Se supponiamo di avere una variabile di tipo data di nome *DataNascita* la seguente istruzione:

```
DataNascita = #12/15/1980#
```

È corretta e indica che la persona è nata il 15 Dicembre del 1980.

VARIABILI

Le variabili sono degli oggetti che contengono i dati elaborati dal programma. La maggior parte delle istruzioni di un programma consiste nell'assegnare il valore di un'espressione a una variabile. Ad ogni variabile viene riservato uno spazio di memoria della dimensione appropriata in base al suo tipo.

DICHIARAZIONE DI UNA VARIABILE

L'istruzione di dichiarazione definisce il nome e il tipo di una variabile, oltre a determinare anche la sua creazione in memoria. Assume la seguente forma:

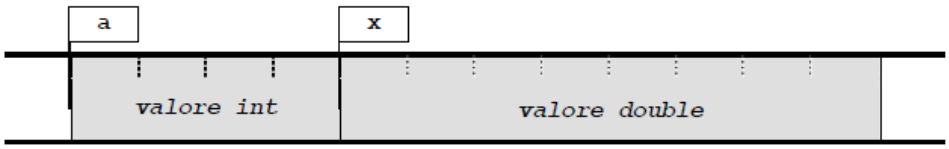
```
DIM nome-var1 AS tipo
```

Le parole in rosso fanno parte della sintassi fissa dell'istruzione di dichiarazione (non possono essere omesse)

Ad esempio, le dichiarazioni:

```
DIM a AS Integer
DIM x AS Double
```

determinano la creazione in memoria di due oggetti, identificati dai nomi `a` e `x`:



Come stabilito dalla sintassi, all'interno della stessa istruzione possono essere dichiarate più variabili; in questo caso, esse appartengono al medesimo tipo. Ad esempio, l'istruzione:

```
DIM a,b,c AS Integer
```

determina la creazione in memoria di tre oggetti di tipo *Integer*.

Il linguaggio non impone requisiti sull'ordine delle dichiarazioni o sulla loro collocazione all'interno del codice sorgente, a parte uno:

La dichiarazione di una variabile deve precedere il suo uso.

Dunque, il seguente codice è scorretto:

```
DIM a AS Integer
a = 1
b = a                    ‘ errore: b non è stata ancora dichiarata!
DIM b AS Integer
```

Mentre il seguente codice è corretto:

```
DIM a AS Integer
a = 1
DIM b AS Integer
b = a
```

Un secondo requisito riguarda l'impossibilità di dichiarare due volte la stessa variabile, o comunque due variabili di tipo diverso ma con lo stesso nome.

Dunque, è un errore scrivere:

```
DIM a, b AS Integer
DIM  x, y AS Double
DIM a AS String          ‘ errore: a è già stata dichiarata!
```

Il tipo delle variabili è in questo caso irrilevante, ciò che ha importanza è il nome.

I precedenti esempi utilizzano spesso dichiarazioni di più variabili nella stessa istruzione. E’ una pratica formalmente corretta, ma sconsigliata, poiché diminuisce la leggibilità del codice. E’ stata più volte utilizzata nel testo soltanto allo scopo di ottenere un codice più compatto.

ASSEGNAZIONE DI UN VALORE AD UNA VARIABILE

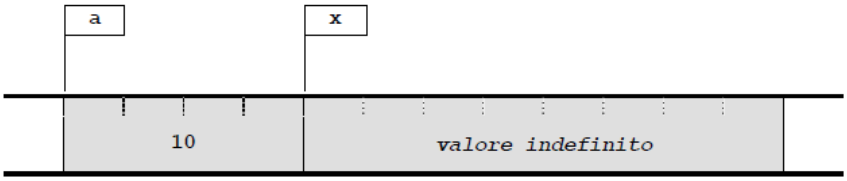
Il modo più comune per modificare il valore di una variabile è mediante un’istruzione di assegnazione. Questa assume sempre la forma:

variabile = espressione

e viene eseguita nel seguente modo:

1) viene innanzitutto valutata *espressione* e cioè calcolato il suo valore. Durante questa fase vengono eseguite tutte le conversioni che sono necessarie in modo che ci sia identità di tipo tra operandi e operatori;
2) il valore prodotto viene posto nella zona di memoria riservata a *variabile*; anche in questo caso se il tipo del valore non coincide con il tipo della variabile avviene una conversione, e cioè il valore viene ricodificato nel tipo della variabile prima di essere memorizzato in essa.

Ad esempio, s’ipotizzi la seguente situazione iniziale:



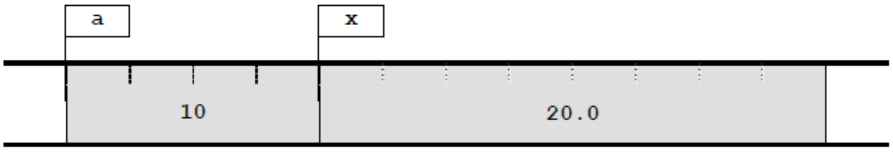
L’assegnazione:

```
x = a + 10
```

produce le seguenti operazioni:

1) viene valutata l’espressione che sta a destra dell’operatore di assegnazione. Questa equivale al valore di *a* più la costante 10.
2) poiché il valore ottenuto è di tipo *integer*, viene promosso al tipo *double*, e cioè il tipo di x;
3) il valore viene memorizzato nella zona di memoria riservata ad x.

Il risultato finale è mostrato in figura:



VALORE INIZIALE DI UNA VARIABILE

Il **valore iniziale** di una variabile è il valore che essa contiene dopo la dichiarazione ma prima che abbia subito un'assegnazione. Ebbene, per le variabili dichiarate in un blocco (attualmente le uniche che stiamo considerando), questo è del tutto casuale. Si dice infatti che la variabile si trova in uno **stato iniziale indefinito**, detto anche **non assegnato**. Per contro, una variabile che abbia subito almeno un'assegnazione si trova in uno stato **definito**.
Il concetto di stato iniziale di una variabile è molto importante poiché il linguaggio proibisce l'utilizzo di una variabile con stato indefinito:

```
DIM x,y,z AS Double
z=10
x=z          ‘ ok: z aveva già un valore, era cioè definita
x=y          ‘ errore: lo stato iniziale di y è indefinito!
```

L'istruzione *x = y* benché appaia corretta, viene segnalata come errata, poiché anche senza eseguire il programma è chiaro che a *z* non è stato assegnato alcun valore e dunque il suo stato non può che essere indefinito. Poiché non avrebbe senso assegnare ad *x* un valore casuale.
In conclusione, la dichiarazione di una variabile determina semplicemente l'allocazione dello spazio in memoria, ma nessun'altra operazione. Prima che vi sia stata almeno un'assegnazione alla variabile, il contenuto di tale spazio è casuale e pertanto la variabile non può essere usata in un'espressione.

INIZIALIZZAZIONE DI UNA VARIABILE

Quello delle variabili non definite è un problema comune della programmazione, per questo motivo VB.NET dà al programmatore la possibilità di specificare il valore iniziale di una variabile già durante la sua dichiarazione. In questo caso la dichiarazione assume la forma:

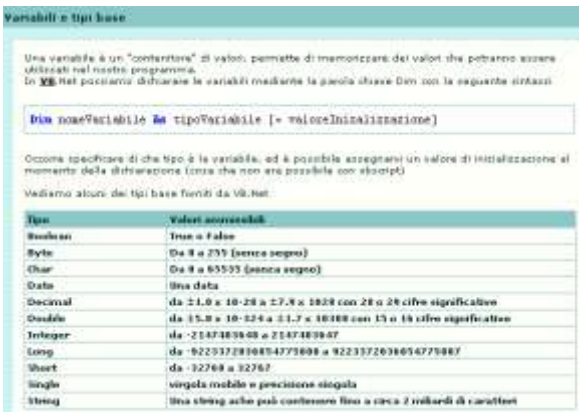
```
DIM nome-variabile AS tipo = valore-iniziale
```

dove valore-iniziale assume anche il nome di inizializzatore della variabile.
Ecco alcuni esempi di dichiarazione con inizializzazione:

```
DIM x AS double = 0
DIM a AS Integer = 1
DIM Trovato AS Boolean = True
DIM Nome AS String = "paperino"
DIM DataNascita AS Date= #12/15/1980#
```

L'inizializzazione di una variabile garantisce che il suo stato iniziale sia definito, ma non è obbligatoria, né sempre desiderabile; essa dipende infatti dall'uso che viene fatto della variabile.

PROSPETTO RIEPILOGATIVO SULLE VARIABILI


Variabili e tipi base

Una variabile è un "contenitore" di valori, permette di memorizzare dei valori che potranno essere utilizzati nel nostro programma.
In VB.Net possiamo dichiarare le variabili mediante la parola chiave Dim con la seguente sintassi

```
Dim nomeVariabile As tipoVariabile [= valoreInizializzazione]
```

Occorre specificare di che tipo è la variabile, ed è possibile assegnarvi un valore di inizializzazione al momento della dichiarazione (cosa che non era possibile con vbscript)

Vediamo alcuni dei tipi base forniti da VB.Net

Tipo	Valori memorizzabili
Boolean	True o False
Byte	Da 0 a 255 (senza segno)
Char	Da 0 a 65535 (senza segno)
Date	Una data
Decimal	da ±1.0 x 10-28 a ±7.9 x 1028 con 28 o 29 cifre significative
Double	da ±5.0 x 10-324 a ±1.7 x 10308 con 15 o 16 cifre significative
Integer	da -2147483648 a 2147483647
Long	da -9223372036854775808 a 9223372036854775807
Short	da -32768 a 32767
Single	virgola mobile e precisione singola
String	Una string può contenere fino a circa 2 miliardi di caratteri

ESPRESSIONI

Un'espressione è rappresentata da:

> una combinazione di uno o più valori (variabili e/o costanti e/o altre espressioni) e di zero o più operatori che operano con essi. Un'espressione denota un valore, il quale appartiene ad un determinato tipo, chiamato tipo dell'espressione.

Data questa definizione, se ne conclude che anche una variabile o una costante designano, da sole, un'espressione. Ecco alcuni esempi:

```
DIM a,b AS Integer
a = 10          ' 10 è un'espressione composta da una sola costante
b = a           ' a è una espressione composta da una sola variabile
b = a + 10      ' a + 10 è un'espressione rappresentata dal valore prodotto dall'operatore + sugli
                  operandi a e 10
```

TIPO DI UN'ESPRESSIONE

Il risultato di un'espressione appartiene sempre ad un determinato tipo, il quale dipende:

> unicamente dai tipi dei valori che vi compaiono e dai tipi degli operatori che operano con essi.

OPERATORI

> Un operatore è un simbolo che designa un'operazione prodotta su uno o più argomenti, detti operandi. Tale operazione produce sempre un risultato di un determinato tipo.

Operatori Aritmetici

Operatore	Descrizione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione decimale
\	Divisione intera
Mod	Modulo
^	Potenza

Operatori Relazionali (di confronto)

Operatore	Descrizione
=	uguale
<>	diverso
>	maggiore
<	minore
>=	maggiore o uguale
<=	minore o uguale

PRECEDENZA DEGLI OPERATORI

La regola di precedenza degli operatori indica, in presenza di un'espressione con più operazioni, l'ordine di esecuzione delle stesse. Consideriamo il seguente codice:

```
DIM a AS Integer=10
DIM b AS Integer= 20
DIM c AS Integer
c = a + b * 10                  // viene eseguito a + (b * 10)
```

L'operatore di moltiplicazione ha la precedenza sull'operatore di somma e dunque, in assenza di parentesi, viene eseguito per primo. Tutti gli operatori del linguaggio sono organizzati secondo un preciso ordine di precedenza.

ESPRESSIONI DI TIPO STRING

Anche sulle stringhe, benché non rappresentino dei valori nel senso che diamo di solito a questo termine, possono essere eseguite delle operazioni.

CONFRONTARE DUE STRINGHE

Due stringhe possono essere confrontate tra loro mediante gli operatori relazionali. In questo caso, due stringhe sono considerate uguali soltanto se hanno la stessa lunghezza e tutti i caratteri corrispondenti sono uguali. Ad esempio:

- "amore" e "amorevole" sono stringhe diverse, poiché la seconda contiene dei caratteri che non sono presenti nella prima;
- "Amore" e "amore" sono stringhe diverse, poiché viene fatta distinzione tra maiuscole e minuscole; (si dice che il confronto è case sensitive, e cioè sensibile al case delle lettere).

Sia il tipo *String* che altri tipi definisco dei metodi di confronto che offrono un maggior controllo al programmatore, come ad esempio quello di poter stabilire se considerare o meno la differenza tra maiuscole e minuscole durante l'operazione di confronto.

CONCATENARE STRINGHE

Due stringhe possono essere concatenate mediante l'operatore di **concatenazione**, rappresentato dal simbolo '&'. Il risultato è rappresentato da una terza stringa contenente i caratteri della seconda attaccati a quelli della prima. Naturalmente, il prodotto della concatenazione di due stringhe può essere a sua volta concatenato con un'altra stringa, e così via.
Come esempio, si consideri il seguente codice:

```
DIM risultato,s1,s2,s3 AS String
s1 = "Stanlio"
s2 = " e "
s3 = "Ollio"
risultato = s1 & s2 & s3
```

s1 & s2 & s3 rappresenta un'espressione di tipo *String*. Il valore di questa espressione è la stringa risultante dalla concatenazione delle variabili *s1*, *s2* e *s3* nell'ordine in cui compaiono nell'espressione, e cioè: "*Stanlio e Ollio*"

Un altro aspetto su cui fare attenzione è che nell'operazione di concatenazione gli spazi vengono trattati né più né meno come gli altri caratteri. Ad esempio, il seguente codice:

```
DIM risultato,s1,s2 AS String
s1 = "ciao "
s2 = "mamma"
risultato = s1 & s2
```

Il valore di questa espressione è la stringa "*ciao mamma*" e non "*ciaomamma*"

ESPRESSIONI DI TIPO BOOLEANO

Le espressioni booleane possono essere **semplici** o **composte**.
Le espressioni semplici esprimono una condizione nella forma:

espressione$_1$ *operatore-relazionale* *espressione*$_2$

laddove i valori delle due espressioni vengono confrontati tra loro mediante uno degli operatori di confronto. Espressioni semplici sono, ad esempio:

```
a > 10      (a + b) = (d + a)      Num Mod 2 = 0
```

Ma come un'espressione qualsiasi può essere composta da un numero arbitrario di sotto espressioni, così un'espressione booleana composta rappresenta la combinazione di più espressioni booleane semplici, connesse dagli **operatori logici condizionali**, o **connettivi**.

Segue l'elenco degli operatori logici condizionali ed il risultato che producono:

OPERATORE LOGICO	CONNETTIVO CORRISPONDENTE	RISULTATO
AND	E	**true** se entrambi gli operandi sono true, **false** altrimenti
OR	O	**true** se uno o entrambi gli operandi sono true, **false** altrimenti
NOT	NON (negazione)	**true** se l'operando è false, **false** se l'operando è true

Un'espressione booleana composta consente la combinazione di più condizioni nella stessa espressione allo scopo di ottenere un codice più compatto ed efficiente. Essa assume la forma:

condizione1 *operatore-logico* *condizione2*

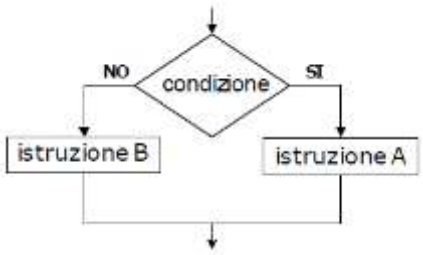
STRUTTURE DI CONTROLLO

SELEZIONE SEMPLICE

Le due principali strutture di controllo presenti nel linguaggio sono la **IF** (**Selezione semplice**) e l'istruzione **SELECT** (**Selezione Multipla**). Mentre la prima consente di eseguire un blocco di istruzioni in relazione ad un test; la seconda consente di discriminare il codice da eseguire in funzione del valore di una variabile di controllo.

La sintassi dell'istruzione **IF** è la seguente:

```
IF condizione Then
       Istruzione A
Else
       Istruzione B
End If
```



La strada dell'Else non è obbligatoria (si può omettere); La condizione può essere vera o falsa: se è vera viene eseguita l'istruzione A altrimenti l'istruzione B.

```
Dim Numero As Integer= 7

If (Numero Mod 2 = 0) Then
     MsgBox("Il numero " & Numero & " è pari!")
Else
     MsgBox("Il numero " & Numero & " è dispari!")
End If
```

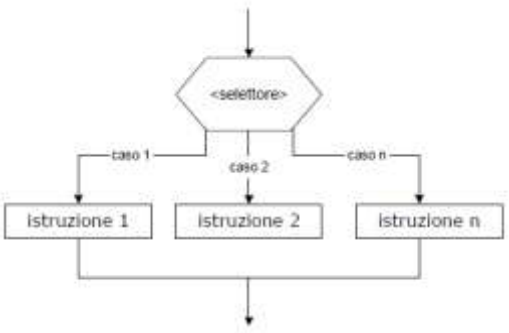
SELEZIONE MULTIPLA

Il costrutto **SELECT** fornisce una particolare implementazione dello schema di selezione multipla. L'istruzione permette di effettuare confronti multipli, anche con range e operatori di confronto. Esso assume la seguente sintassi, qui presentata in una forma semplificata:

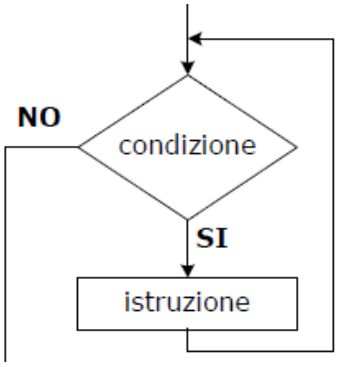
```
SELECT CASE TestExpression

       CASE ExpressionList
              ???????

       CASE ELSE
              ???????
END SELECT
```

All'interno dell'*ExpressionList* è possibile utilizzare le parole chiave **TO** e **IS** per effettuare dei confronti e delimitare dei range, come illustrato nell'esempio:

```
Dim Numero As Integer= 7

Select Case Numero
    Case 1 To 5
        MsgBox("Compreso tra 1 e 5")
    Case 6, 7, 8
        MsgBox("6 , 7 o 8")
    Case 9 To 10
        MsgBox("Compreso tra 9 e 10")
    Case Else
        MsgBox("Non Compreso tra 1 e 10")
End Select
```



STRUTTURA ITERATIVA - WHILE



Il ciclo **WHILE** viene eseguito nel seguente modo:

a) viene valutata la condizione nel rombo (chiamata anche condizione di controllo del ciclo);

b) se la condizione è vera, viene eseguita l'istruzione immediatamente successiva (chiamata corpo del ciclo: nel disegno il rettangolo di nome Istruzione) , dopodiché si ritorna al punto a;

c) se invece la condizione è falsa il ciclo termina.

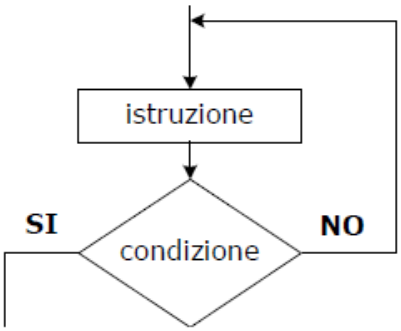
Il costrutto *WHILE* presenta la seguente sintassi:

```
Do WHILE condizione
       ???????
Loop
```

Il ciclo WHILE si dice anche che cicla per *Vero* ed esce per *Falso*.

```
Dim i As Integer
i = 1
Do While (i <= 10)
     MsgBox("i = " & i)

     i = i + 1
Loop
```

STRUTTURA ITERATIVA - REPEAT



Il ciclo **REPEAT** viene eseguito nel seguente modo:

a) viene eseguita l'istruzione (blocco rettangolare) chiamata corpo del ciclo
b) viene valutata la condizione nel rombo (chiamata anche condizione di controllo del ciclo);
c) se la condizione è falsa si ritorna al punto a;
d) se invece la condizione è vera il ciclo termina.

Il costrutto *REPEAT* presenta la seguente sintassi:

```
Do
       ???????
Loop UNTIL condizione
```

Il ciclo REPEAT si dice anche che cicla per *Falso* ed esce per *Vero*.

```
Dim i As Integer

i = 0
Do
     i = i + 1
     MsgBox("i = " & i)

Loop Until (i = 10)
```

STRUTTURA ITERATIVA - FOR

Gestisce un'iterazione controllata da contatore. Usato quando conosciamo a priori il numero di iterazioni che andremo ad eseguire (Ciclo definito)

```
For contatore = valoreIniziale To valoreFinale [Step incremento]
  Azioni
Next
```

La parola chiave Step, può essere un intero (positivo o negativo) e rappresenta il valore di incremento della variabile di controllo del ciclo. Se omessa vale 1. Invece 1 e 30 sono il valore iniziale e finale: possono anche essere delle espressioni.

```
Dim i As Integer
For i = 1 To 30 Step 1
     MsgBox("i = " & i)
Next
```