Richiami d'informatica – Visual Basic For Application

Dispensa redatta dall'Ing. Francesco Zammori
francesco.zammori@unipr.it
Ad uso degli studenti del corso di Gestione dell'informazione Aziendale A.A. 2018/2019
Corso di laurea magistrale in Ingegneria Gestionale, Università di Parma

1. Tipi dati, variabili e costanti

Ogni volta che si scrive del codice (in un qualsivoglia linguaggio di programmazione), è necessario gestire una mole di dati di diversa natura:

- numeri;
- stringhe, ossia dati alfanumerici (composto da numeri e lettere);
- data e ora;
- valori monetari;
- informazioni booleane (di tipo True oppure False);
- tipi di dati definiti dall'utente;
- tipi di dati strutturati (array monodimensionali e pluridimensionali). Gli array bidimensionali sono le matrici, quelli monodimensionali sono detti anche vettori;

Per gestire questi dati è necessario memorizzarli in opportune variabili.

Una variabile VBA è identificata da tre elementi:

- un nome;
- un valore;
- un tipo.

Il nome può essere scelto a piacere, ma dovrebbe "evocare" il contenuto assegnato alla variabile. La scelta non è totalmente libera dato che: *(i)* il nome non può essere uguale a una keyword del linguaggio (parole riservate), *(ii)* non può avere più di 255 caratteri, *(iii)* non può avere spazi o segni di punteggiatura e *(iv)* non può iniziare con un numero. I caratteri possono essere scritti indifferentemente in minuscolo o in minuscolo dato che VBA è "case insensitive".

Per dichiarare una variabile si utilizza l'istruzione **Dim**:

```
Dim Nome As Tipo
```

In contrapposizione alle variabili, i dati che non saranno soggetti a cambiamenti nel corso di esecuzione dell'applicazione, sono denominate costanti.
Per dichiarare una costante ci sono due tecniche:

```
Const Nome = Valore
```

oppure

```
Const Nome As Tipo = Valore
```

Questa forma permette di rendere esplicita la dichiarazione del tipo per una costante. La prima forma, anch'essa corretta, può essere utilizzata in virtù del fatto che i tipi delle costanti sono dedotti in base al valore assegnato.
Come visto, sia che venga dichiarata una variabile o una costante è possibile definirne il tipo, ossia specificare la tipologia di dati che verrà assegnata a tale variabile. I tipi disponibili sono riassunti nella seguente tabella.

Tipi disponibili in VBA

Tipo	Descrizione	Dati rappresentati
Boolean	Memorizza valori dell'Algebra di Boole	True oppure False
Byte	Memorizza valori naturali	Intervallo di valori compresi tra 0 e 255
Currency	Memorizza valori monetari compresi 4 cifre decimali	Intervallo compreso tra -922 a +922
Date	Memorizza informazioni circa data e ora	Dal 1° gennaio 100 al 31 dicembre 9999
Double	Memorizza valori decimali a precisione doppia	Intervallo numeri negativi da -1,797679E308 a -4,94065E-324 Intervallo numeri positivi da 4,04065E-324 A 1,79679E308
Integer	Valori naturali con segno	Intervallo -32.768 a + 32.767
Long	Valori naturali con segno	Intervallo -2.147.... a + 2.147....
Object	Memorizza un riferimento all'oggetto	Dipende dalla natura dell'oggetto
Single	Valori decimali a singola precisione	Intervallo numeri negativi: da -3,4028... a -1,4012... Intervallo numeri positivi da 1,4012... a 3,4028...
String	Memorizza stringhe alfa numeriche	Memorizza qualsiasi stringa di caratteri
Variant	Memorizza qualsiasi tipo	Il dato rappresentato dipende dal tipo rappresentato.

Una volta definite (dichiarate le variabili) è possibile assegnare ad esse dei valori. Ad esempio:

```
Dim S As String
Dim i As Integer, j As integer 'è possibile dichiarare più variabili sulla stessa riga
[…]
S = “Una Stringa” 'le stringhe vanno racchiuse tra doppi apici
i = 10
j = 5
```

Chiudiamo questo paragrafo ricordando che:

- Quando definiamo una variabile in un modulo (come vedremo si tratta di uno spazio riservato, una pagina, in cui scrivere il codice), è possibile far precedere a Dim la dicitura **Public** o **Private**. Nel primo caso la variabile ha visibilità globale e potrà essere richiamata anche da codice scritto in altri moduli. Nel secondo caso, la variabile sarà privata e, pertanto, sarà richiamabile solo all’interno del modulo in cui è stata definita.
- È possibile iniziare un modulo con la direttiva **Option Explicit**; in questo caso si rende obbligatoria la dichiarazione delle variabili. In caso contrario è possibile assegnare un valore ad una variabile anche senza averla precedentemente dichiarata. In particolare, ad una variabile non dichiarata con l'istruzione Dim, verrà automaticamente assegnato il tipo universale Variant, con ovvio spreco di maggior memoria per l'allocazione dell'informazione.

2. Tipi personalizzati e Array

Vediamo ora come fare a dichiarare tipi di dati più complessi, ossia i tipi definiti dall'utente e gli array. I dati definiti dall'utente sono una sorta di contenitore che consente di raggruppare in un'unica variabile dati di tipologia anche differente. Supponiamo ad esempio di voler creare un archivio in cui registrare nome, cognome e credito elargito ad alcuni nostri clienti. In questo caso, dato che l'archivio di ciascun cliente dovrà contenere tre dati è utile definire un tipo personalizzato. Per farlo è necessario racchiudere la definizione delle variabili di nostro interesse all'interno dell'istruzione **Type … End Type,** come indicato nel seguente schema sintattico.

```
Type Nome_Tipo
        Nome_Prima_Variabile As Tipo
        …
        Nome Ultima_Variabile As Tipo
End Type
```

Per cui, nel caso d'esempio si potrebbe scrivere qualcosa del tipo:

```
Type archivio
  cognome As String*30
  nome As String*30
  credito As Double
End Type
```

Si noti che l'indentazione del codice non è obbligatoria, ma rende il codice più chiaro e leggibile. All'interno dell'istruzione type, come anche del resto, nelle istruzioni di dichiarazione Dim, le stringhe a lunghezza fissa sono dichiarate così usando l'asterisco seguito dal numero di caratteri desiderati.

È importante notare che una volta definito un tipo, in memoria non è stato creato nessun riferimento. Al fine di creare un riferimento in memoria, dovremmo dichiarare questo tipo di dato con l'istruzione Dim. Ad esempio, nel caso d'esempio bisognerà scrivere qualcosa del genere:

```
Dim Cliente As Archivio
[…]
Cliente.cognome = "Rossi"
Cliente.nome = "Marco"
Cliente.credito = 302.2
[…]
```

Si noti che, per assegnare un valore ad ogni variabile contenuta nel tipo definito dall'utente è necessario far seguire al nome della variabile l'operatore punto '.' seguito dal nome della variabile interna:

```
Nome_tipo.NomeVariabile = Valore
```

Al momento abbiamo creato un archivio composto da un solo cliente. Cosa questa abbastanza strana. Di clienti ne avremo infatti ben più di uno. È certamente possibile creare tante variabili di tipo archivio quanti sono i nostri clienti, ma se ci limitassimo a questo, le variabili resterebbero "sparpagliate" e sarebbe difficile gestirle in maniera consistente. Sarebbe molto meglio raggrupparle in una struttura unica, un vero e proprio archivio composto di dati di più clienti. Per farlo possiamo creare un array (monodimensionale) e popolarlo con i nostri clienti.

Esattamente come per le variabili semplici, per creare delle variabili strutturate, per le quali a ciascun elemento si accede con uno o più indici interi positivi, si procede così:

```
Dim Nome_Array(dimensione) As Tipo
```

dove la dimensione specificata tra parentesi tonda definisce il numero di elementi che costituiscono il nostro array. È possibile specificare sia l'indice inferiore sia l'indice superiore, oppure solo l'indice superiore. In quest'ultimo caso il primo elemento avrà indice zero (di default gli array sono indicizzati a partire da zero. Ad esempio:

```
Dim Array(1 to 10) As Integer 'Un vettore contenente 10 elementi di tipo intero
[…]
Dim Array_Bis(10) As Integer 'Un vettore contenente 11 elementi di tipo intero, indicizzato da 0
[…]
Array(1) = 10 ' Assegna il valore 10 al primo elemento dell'array
[…]
I = Array(1) ' Assegna alla variabile i il valore del primo elemento dell'array
```

In maniera del tutto simile, per definire una matrice è necessario definire le dimensioni delle righe e delle colonne.

```
 Dim Array(1 to 10, 1 to 10) As Integer 'Una matrice 10x10 di interi
[…]
Array(1,1) = 1 'Si assegna il valore di 1 al primo elemento della matrice
[…]
```

Come vedremo in esempi successivi, quando si lavora con array sono utili le seguenti funzioni:

- **Ubound** – Restituisce l'indice superiore dell'array
- **Lbound** – Restituisce l'indice inferiore dell'array

Osserviamo infine che, quando si definisce un array, non è obbligatorio definire la sua dimensione. È possibile farlo in un secondo momento (ad esempio quando sapremo di che spazio di memoria avremo bisogno) prima di iniziare ad assegnare valori al nostro array.
Per definire la dimensione dell'array si usa l'istruzione **Redim** (ridimensiona) come mostrato di seguito:

```
Dim Array() As Integer 'Un array di dimensioni indefinite contenente interi
[…]
Redim Array(1 to 10) ' Definita la dimensione dell'array lo si può utilizzare
[…]
Array(1) = 10
[…]
```

È anche possibile ridimensionare più volte uno stesso array. Tuttavia, così facendo, ogni volta in cui utilizziamo l'istruzione Redim, perdiamo tutti i valori assegnati al nostro array. Se invece volessimo conservare tali valori dovremmo scrivere **Redim Preserve**.

Tornando all'esempio dell'archivio, potremmo allora scrivere qualcosa di questo genere:

```
Dim Elenco(1 To 10) As Archivio
Dim Cliente1 As Archivio
Dim Cliente2 As Archivio
[…]
Dim Cliente10 As Archivio
Cliente1.Nome = "Francesco"
Cliente1.Cognome = "Verdi"
Cliente1.Credico = 100
[…]
Cliente10.Credito = 50
Elenco(1) = Cliente1
[…]
Elenco(10) = Cliente10
```

3. Funzioni e Procedure

Procedure e funzioni sono il mattone di base di ogni linguaggio di programmazione. Sono due elementi molto simili, dato che, entrambe servono a richiamare una serie di istruzioni che devono essere utilizzate più volte.
In alcuni linguaggi non esiste neppure una differenza tra di loro, viceversa, nei linguaggi più strutturati (come VBA), la differenza è la seguente:

- Una funzione opera su dati d'ingresso e restituisce un valore in output;
- Una procedura esegue alcuni compiti, può ricevere alcune variabili di input (necessarie a svolgere talune operazioni), ma non restituisce nessun valore.

Facciamo subito un esempio:

```
Public Sub Show_Message(Message As String)
   MsgBox Message, vbOKOnly, "A first Example"
End Sub
```

Show_Message è una procedura che riceve in input una variabile di tipo stringa che verrà mostrata a video mediante il comando predefinito MsgBox, comando che a sua volta è una procedura predefinita contenuta nelle librerie che VBA ci mette a disposizione.

In particolare, la sintassi di una procedura è la seguente:

```
Public|Private Sub Procedure_Name (Input_1 As Type … Input_N As Type)
        [Statements]
[Exit Sub]
        [Statements]
End Sub
```

In pratica:

- Il codice che vogliamo eseguire deve essere racchiuso tra le keyword **Sub** ed **End Sub**;
- La keyword Sub (preceduta da Public o da Private in funzione del grado di visibilità che si vuol dare alla procedura) deve essere seguita dal nome che vogliamo attribuire alla nostra procedura;
- Tra parentesi tonde va messa la lista degli input che dobbiamo fornire alla procedura. Si noti che è necessario definire il tipo di tali input.
- All'interno della procedura può essere usata l'istruzione **Exit Sub** che, se raggiunta blocca l'esecuzione del codice restante (in pratica tutte le righe di codice poste al di sotto di Exit Sub non verranno considerate).

Tornando al caso d'esempio, abbiamo creato una semplicissima procedura denominata Show_Message che richiede in input un messaggio di testo. Per richiamare tale procedura è necessario usare la keyword **Call** (chiama) seguita dal nome della procedura corredata dei necessari parametri di input. Ad esempio, scrivendo:

```
Call Show_Message("Ciao")
```

apparirà a video la scritta "Ciao".
Si noti che, essendo di tipo stringa, la variabile di input va passata tra virgolette doppie.

L'esempio seguente è invece relativo ad una funzione. Come detto, la struttura è molto simile alla precedente, ma in questo caso la keyword Sub è sostituita con la keyword **Function** e, a differenza di prima, dopo la lista dei parametri di input c'è un'ulteriore dichiarazione di variabile. Si tratta della variabile che verrà restituita in uscita e che, per default assume lo stesso nome dato alla funzione.

Per completezza, si riporta la sintassi completa, simile alla precedente:

```
Public|Private Function Function_Name (Input1 As Type … InputN As Type) As Type
        [Statements]
        [Exit Function]
        [Statements]
End Function
```

Vediamo subito un esempio:

```
Public Function Radice_N(N As Double, Optional R As Double = 2) As Double
  Radice_N = N ^ (1 / R)
End Function
```

In questo caso, abbiamo definito una funzione che restituisce la radice ennesima di un numero; radice che è restituita come variabile di tipo Double (numero con virgola).
La funzione ha bisogno di due parametri: il numero a cui applicare il parametro radice e la molteplicità della radice stessa. Si noti l'utilizzo della keyword **Optional** che rende opzionale l'immissione della seconda variabile di input; se omessa, tale variabile sarà pari a due. Per cui se digitiamo

```
Dim A As Double
A = Radice_N(9)
```

A assumerà il valore di 3, dato che di default viene eseguita la radice quadrata.
Viceversa, scrivendo:

```
A = Radice_N(8,3)
```

la variabile A assumerà un valore pari a 2.
Si noti che a differenza di prima non è possibile chiamare una funzione, ma è necessario assegnarla ad una variabile. A differenza di una procedura una funzione non esegue una porzione di codice, ma genera un valore; tale valore va necessariamente associato ad una variabile di tipo compatibile con quello della funzione.
Concludiamo questa sezione con un'ultima osservazione. Di default quando definiamo la lista dei parametri di una funzione, tali parametri sono passati per riferimento (**ByRef**); pertanto ciascuna modifica su di essi si ripercuote sulla variabile passate in input. Viceversa, se volessimo evitare tale comportamento dovremmo passare le variabili di input per valore mediante l'istruzione **ByVal**. In questo modo, modifiche apportate alle variabili di input internamente ad una funzione non modificano il valore delle variabili all'esterno della funzione. Di seguito è riportato un semplice esempio:

```
Public Sub Reference_Example()
Dim My_Var As Integer
Dim New_Var As Integer
  My_Var = 10
  New_Var = Do_Not_Change(My_Var) ' Funzione alla quale il parametro viene passato ByVal
  Debug.Print New_Var ' Mostra a video il valore della variabile
  Debug.Print My_Var
  New_Var = Change(My_Var) ' Funzione alla quale il parametro viene passato ByRef
  Debug.Print New_Var
  Debug.Print My_Var
End Sub
```

```
Private Function Change(ByRef N As Integer) As Integer
   N = N * 3
   Change = N
End Function

Private Function Do_Not_Change(ByVal M As Integer) As Integer
   M = M * 2
   Do_Not_Change = M
End Function
```

Chiamando la procedura Reference_Example si ottengono i seguenti valori: 20; 10; 30; 30. La spiegazione di tale risultato è lasciata, come esercizio, al lettore.

4. Istruzioni condizionali e cicli

Spesso è necessario fare delle scelte, ossia è necessario eseguire una parte di codice a fronte del verificarsi di una o più condizioni logiche.

If … Then … Else e Select Case, sono le principali strutture sintattiche usate per effettuare delle scelte, all'interno di un codice.

La sintassi della struttura IF … Then è riportata di seguito:

```
If condition Then
        [statements]
[Else]
        [statements]
End If
```

La sintassi della struttura Select Case è riportata di seguito:

```
Select Case Value
        Case possible Value [To possible Value]
            [statements]]
        [Case possible Value [To possible Value]
            [statements]]
        [Case Else
            [statements]]
End Select
```

Vediamo un semplice esempio:

```
Dim N As Integer = 8
Select Case N
   Case 1 To 5 ' Si esegue se il numero N è compreso tra 1 e 5
      Debug.Print("Between 1 And 5, inclusive")
   Case 6, 8 ' Si esegue se il numero N è pari a 6 o ad 8
      Debug.Print("Equal to 6 Or 8")
   Case 9 To 10 ' Si esegue se il numero N è compreso tra 9 e 10
      Debug.Print("Between 9 And 10, inclusive ")
   Case Else ' Si esegue in tutti gli altri casi (ad esempio se N = 7)
      Debug.Print("7 Or (Not between 1 And 10, inclusive)")
End Select
```

Si noti che, avendo posto N = 8, l'unica istruzione che verrà eseguita sarà la seguente:

```
   Debug.Print("Equal to 6 or 8")
```

Altre volte è invece necessario ripetere una stessa operazione ripetutamente; per farlo possiamo usare i cicli **Do … Loop** o **For … Next** la cui sintassi è mostrata di seguito.

```
Do [{While | Until} condition]
        [statements]
        [Exit Do]
        [statements]
Loop
```

In pratica la prima riga specifica una condizione. Se vera tale condizione fa partire il ciclo e tutto il codice compreso fino alla keyword Loop verrà eseguito.
Nel caso in cui si usi la keyword **While** L'esecuzione di tale codice viene re-iterata sino a che la condizione si mantiene vera. Viceversa, usando **Until** l'esecuzione s'interrompe non appena la condizione specificata diventa vera. In entrambi i casi è possibile interrompere l'esecuzione anticipatamente usando l'espressione **Exit Do**.
Esiste anche una formulazione alternativa, come mostrato di seguito:

```
Do [statements]
        [Exit Do]
        [statements]
Loop [{While | Until} condition]
```

In questo caso la condizione viene valutata solo alla fine per cui, indipendentemente dal fatto che la condizione sia vera o falsa, le istruzioni contenute tra Do e Loop vengono eseguite almeno una volta.

Esiste anche una sintassi semplificata, ma meno flessibile che prevede l'uso delle keyword **While … Wend**, come mostrato di seguito:

```
While condition
        [statements]
Wend
```

In questo caso, trattandosi di un ciclo While, il ciclo prosegue sino a che la condizione iniziale si mantiene vera.

Consideriamo un semplice esempio:

```
Dim Counter As Integer
Counter = 0
While Counter < 20
        Counter = Counter + 1
Wend
Debug.Print "Alla fine del ciclo il contatore vale: " & Counter
```

In questo caso la variabile contatore (Counter) viene progressivamente incrementata, sino a che il suo valore non diventa pari a 20. Si noti inoltre che, avendo inizializzato tale variabile ad un valore pari a 0, il ciclo verrà eseguito esattamente 20 volte. Ovviamente, in uscita la variabile varrà 20 e, a video si leggerà il seguente messaggio "Alla fine del ciclo il contatore vale: 20".

Infine, la forma **Do … Loop** permette di ripetere un certo numero d'istruzioni per un numero predefinito di volte.
La sintassi è la seguente:

```
For counter = start To end [Step]
        [statements]
        [Exit For]
        [statements]
Next [counter]
```

In questo caso, tutte le istruzioni tra **For … Next** sono eseguite un numero di volte predefinito, a meno che non venga attivata l'istruzione **Exit For** (che termina l'esecuzione del ciclo). Il contatore di ciclo (counter) inizia dal valore start e ad ogni iterazione è incrementato di un valore pari a step. Tale valore (opzionale) di default è pari a 1, ma può avere qualsiasi valore intero anche negativo. Il ciclo s'interrompe quando il contatore supera il valore end. Quindi, il numero totale di esecuzioni è pari a pari a (end – start)/step.
Detto questo, l'esempio precedente potrebbe, in cui la variabile Counter veniva incrementata da 0 a 20 con un ciclo While Wend, potrebbe essere riscritto nel modo seguente utilizzando un ciclo For:

```
Dim Counter As Integer
Counter = 0
For i = 1 To 20
   Counter = Counter + 1
Next i
```

5. Esempi riassuntivi[1]

5.1 Fattoriale

Il primo esempio riguarda una funziona che calcola il fattoriale di un numero.
Ricordiamo che il fattoriale rappresenta il numero di modi con cui possono essere permutati *n* oggetti. In particolare, si ha che:

$$n! = n \times (n - 1) \times (n-2) \times \ldots \times 1.$$

Inoltre, vale:

$$0! = 1! = 1.$$

Tale funzione si avvale di un'istruzione condizionale di tipo If … Then per valutare la correttezza della variabile d'input e di un ciclo For … Next di tipo inverso (con step negativo pari a -1) per calcolare il fattoriale.

```
Public Function Fattoriale(Num As Integer) As Single
Dim i As Integer
  If Num <= 1 Then
    Fattoriale = 1
  Else
    Fattoriale = Num
  End If
  For i = (Num - 1) To 1 Step -1
    Fattoriale = Fattoriale * i 'Si noti che la variabile contatore i è usata attivamente nel ciclo
  Next i
End Function
```

5.2. Coefficiente binomiale

La seconda funzione che presentiamo richiama la funzione precedente (la funzione fattoriale) per calcolare il coefficiente binomiale (n, k) che esprime il numero di gruppi di *k* oggetti (senza ripetizione e senza ordinamento) che possono essere formati partendo da un insieme di *n* oggetti. In particolare, il coefficiente binomiale è definito come segue:

$$(n, k) = n!/(n! \times (n – k)!)$$

```
Public Function CB(N As Integer, k As Integer) As Single
Dim N1 As Double, k1 As Double, nk1 As Double
Dim S As Integer
  If N > k Then
    N1 = Fattoriale(N)
```

[1] *Si osserva che, per poter essere eseguite tutte le funzioni successive devono essere poste all'interno di un modulo di VBA.*

```
    k1 = Fattoriale(k)
    S = N - k
    nk1 = Fattoriale(S)
    CB = N1 / (k1 * nk1)
  Else
    CB = 1
  End If
End Function
```

5.3. MCD

Il terzo esempio riguarda una funzione che calcola il Massimo Comun Denominatore di due numeri naturali N_1 e N_2. L'algoritmo utilizzato (non è il solo) è il seguente:

- Dividiamo N_1 per N_2; sia R_1 il resto di tale divisione. Se $R_1 = 0$, allora N_2 è un divisore intero di N_1 e, ovviamente, MCD = N_2. Stop.
- Viceversa, se $R_1 > 0$, se R_1 dovesse essere un divisore di N_2, lo sarà anche di N_1 (essendo il resto della divisione di tali due valori). Pertanto, si divide N_2 per R_1 e, detto R_2 il nuovo resto, se $R_2 = 0$, allora MCD = R1. Stop.
- In caso contrario si prosegue dividendo R_1 per R_2 e così via sino a quando si ottiene un resto R_i (alla i-esima iterazione) pari a zero oppure pari a uno.

Vista la peculiarità di tale algoritmo (per il quale sappiamo definire una condizione di terminazione, ma per il quale non possiamo conoscere a priori il numero di cicli necessari), la funzione si avvale di un ciclo di tipo While … Loop.

```
Public Function MCD(N1 As Integer, N2 As Integer) As Integer
Dim Dividendo As Integer, Divisore As Integer, Resto As Integer
    If N1 > N2 Then
      Dividendo = N1
      Divisore = N2
  Else
      Dividendo = N2
      Divisore = N1
  End If
  Resto = Rst(Dividendo, Divisore) 'funzione privata, vedi di seguito
  If Resto = 0 Then
      MCD = Divisore
      Exit Function
  Else
      Do While Resto > 1 'While prosegue sino a che la condizione si mantiene vera
        Dividendo = Divisore
        Divisore = Resto
```

```
        Resto = Rst(Dividendo, Divisore)
    Loop
  End If
    If Resto = 0 Then
    MCD = Divisore
  Else
    MCD = 1
  End If
End Function
Private Function Rst(a As Integer, b As Integer) As Integer
  Rst = a - b * (a \ b) ' \ è la divisione intera
End Function
```

5.4. Conversione di Base (I)

Il quarto esempio riguarda una funzione che converte un numero N_b espresso in una base *b* (da 2 a 9) nella sua rappresentazione N_{10} in base 10. L'algoritmo è molto semplice.

- Si legge la cifra più a destra di N_b,
- Detto $n_{1,b}$ tale valore, $n_{1,10}$ in base dieci sarà pari a $[(b^0) \times n_1]$.
- Si prosegue con la seconda cifra più a destra, detta $n_{2,b}$, e la si converte nel modo seguente: $n_{2,10} = [(b^1) \times n_2]$.
- Si prosegue così sino all'ultima cifra (quella più a sinistra).
- N10 si ottiene andando a sommare tutte le conversioni intermedie: $N_{10} = n_{1,10} + n_{2,10} + \ldots$

Supponiamo di dover convertire 101 da base 2 a base 10. Si ha: $2^0 \times 1 + 2^1 \times 0 + 2^2 \times 1 = 5$.
Si noti che al fine di leggere ogni singola cifra, il valore numerico passato in input è convertito in stringa, con la funzione di **type_casting Ctsr()**.
Si usano inoltre le seguenti funzioni standard:

- *Len(Stringa)* – Restituisce la lunghezza (numero di caratteri della stringa),
- *Right(Stringa, n)* – Restituisce gli enne caratteri più a destra della stringa,
- *Mid(Stringa, p, n)* – restituisce gli enne caratteri della stringa a partire dal carattere in posizione *p*.

```
Public Function CB_10(N As Integer, Optional b = 2) As Integer
Dim i As Integer, Counter As Integer
Dim S As String, Ch As String
  CB_10 = 0
  If b > 9 Or b <= 1 Then Exit Function
   S = CStr(N)  'Type Casting da intero a stringa
  Counter = 0
  Do While Len(S) > 0 'Qui leggiamo la lunghezza della stringa
    Ch = Right(N, 1) 'Si parte sempre dall'ultima cifra, ossia il carattere più a destra di S
    If Int(Ch) >= b Then 'Non possono esserci valori maggiori o uguali della base
```

```
        CB_10 = 0
        Exit Function
    End If
    CB_10 = CB_10 + Int(Ch) * b ^ Counter 'La si eleva
    N = Mid(N, 1, Len(N) - 1) 'Si modifica la stringa, in pratica si toglie la cifra appena letta
    Counter = Counter + 1
  Loop
End Function
```

Si osservi come sia possibile risolvere tale problema anche utilizzando in ciclo For … Next fatto su tutte le cifre, o meglio i caratteri della stringa numerica N. La conversione del ciclo While in ciclo For è lasciata per esercizio al lettore.

5.5. Conversione di Base (II)

Il quinto esempio riguarda una funzione che permette di convertire un numero N_{10} espresso in base 10 nella sua rappresentazione N_b in base $b < 10$.
L'algoritmo è il seguente:

- si divida N_{10} per b, e siano Q_1 il quoziente e R_1 il resto di tale divisione.
    - Se $R_1 < b$ si vada allo step successivo.
    - Altrimenti si divida R_1 per b e siano Q_2 il quoziente e R_2 il resto di tale divisione
    - …
    - Si proceda così sino a che il resto dell'i-esima divisione risulti minore di b.
- Si concatenino i resti delle divisioni con l 'ultimo quoziente ottenuto (partendo dall'ultimo resto e risalendo sino al primo). Tale concatenazione fornisce il numero in base 10 cercato.
- In pratica, detto n il numero di divisioni effettuate si ha $N_b = Q_nR_nR_{(n-1)} \ldots R_1$

Ad esempio, se volessimo convertire 6 in base 2, avremmo:

- 6/2 = 3, resto = 0;
- 3/2 = 1, resto 1;
- Si ottiene 110 (ultimo quoziente, combinato con i due resti)

Anche in questo caso si usa lo stratagemma di rappresentare il numero come stringa; in questo modo risulta più semplice concatenare una cifra, ossia un carattere, alla volta.

```
Public Function Ten_CB(N As Double, Optional b = 2) As Single
Dim S As String
Dim R As Integer, Q As Integer
  S = ""
  N = Int(Round(N, 0))
  Ten_CB = 0
  If b >= 10 Or b <= 1 Then Exit Function
  Q = b + 100 'Un valore grande a caso, tanto per far partire il ciclo
  Do While Q >= b
    Q = N \ b 'Divisione Intera
```

```
    R = N - b * Q 'Il resto
    S = Str(R) & S 'Si osservi come il valore calcolato si “appende” alla sinistra della stringa
    S = Trim(S)
    N = Q
  Loop
        S = Str(Q) & S
  Ten_CB = CSng(S) 'Riconvertiamo la stringa in numero – in Single
End Function
```

5.6. Matrice Identità

Il sesto esempio concerne una funzione che genera una matrice identità, ed una procedura che provvede a stampare a video la matrice precedentemente ottenuta. Tale esempio, semplice nella sua linearità, mostra una cosa interessante. Generalmente, infatti, una funzione restituisce un solo valore. In questo caso, invece, la nostra funzione deve restituire una matrice. Come si può fare? In VBA è possibile creare funzioni che restituiscono più valori definendo come Variant la variabile restituita in output.

```
Public Function MatriceIdentità(NN As Integer) As Variant
Dim i() As Variant 'La nostra matrice identità
Dim R As Integer, C As Long
  ReDim i(1 To NN, 1 To NN) 'Ridimensioniamo la matrice
  N = 0
  For R = 1 To NN 'Doppio ciclo per compilare la matrice
    For C = 1 To NN
      i(R, C) = 0
      If R = C Then i(R, C) = 1 'Uno se r=c, 0 altrimenti
    Next C
  Next R
  MatriceIdentità = i 'Si noti questo assegnamento diretto, vale solo tra array di variant!!!
End Function
Public Sub PrintI(Optional NN As Integer = 4)
Dim R As Integer, C As Integer
Dim i() As Variant
Dim S As String
  i = MatriceIdentità(NN) 'Richiama la funzione precedentemente definita per creare i()
    For R = 1 To UBound(i, 1) 'Si noti l’uso di Ubound per definire l’estremo del ciclo
    For C = 1 To UBound(i, 2)
      S = S & " " & i(R, C)
    Next C
```

```
    S = S & vbNewLine 'Infondo alla riga mettiamo il carattere di “a capo”
  Next R
    Debug.Print S
End Sub
```

5.7. Array di variabili definite dall’utente

Il settimo esempio illustra la creazione dell’elenco descritto al paragrafo 2. In questo caso si fa uso di un tipo definite dall’utente e di un vettore atto a contenere tale tipo di dati.

```
Type Archivio
  Nome As String
  Cognome As String
  Credito As Currency
End Type
Public File(1 To 10) As Archivio 'Il vettore contenente gli archivi dei singoli clienti
Private Sub Fill_File() 'La funzione che compila l’archivio
Dim F As Archivio 'Variabile di tipo definito dall’utente
Dim i As Integer, j As Integer, k As Integer
Dim N As String, C As String
  For i = 1 To 10 'I valori vengono messi in maniera casuale
    N = ""
    C = ""
    k = Application.WorksheetFunction.RandBetween(1, 10)
      For j = 1 To k 'Chr converte un numero (codice Ascii) nel simbolo corrispondente
        N = N & Chr(Application.WorksheetFunction.RandBetween(65, 90))
        C = C & Chr(Application.WorksheetFunction.RandBetween(65, 90))
      Next j
    F.Cognome = C
    F.Nome = N
    F.Credito = Application.WorksheetFunction.RandBetween(0, 1000)
    File(i) = F
  Next i
End Sub
Public Sub Print_file()
Dim i As Integer, k As Integer
Dim S As String
  Call Fill_File
```

```
  For i = 1 To 10
    S = File(i).Nome & ", " & File(i).Cognome & ", Credito: " & File(i).Credito
    Debug.Print S
    Debug.Print ""
  Next i
End Sub
```

5.8. Un primo esempio di programma – Vincere alla roulette

Di seguito viene presentato un semplice programma che simula l'unico comportamento che consentirebbe di vincere (avendo un capitale pressoché illimitato) alla roulette. In pratica il giocatore punta su un numero, se vince si ritira altrimenti gioca nuovamente puntando una somma che gli consenta di coprire la perdita precedente e di mettere da parte qualcosa. Il giocatore va avanti così (incrementando la punta) sino a quando non scommette sul numero vincente.

Il programma simula un numero enne di giocate successive e, alla fine restituisce il numero medio di giocate fatte prima di vincere, la vincita media e il capitale (le puntate) mediamente investito. Il programma non commentato, la cui comprensione è lasciata come esercizio al lettore è mostrato di seguito.

```
'Le variabili globali dove salviamo i dati generati da una simulazione della giocata
Public Win As Double, Net_Win As Double, Tot_Loss As Double, Tot_Bet As Double
Public N_Bet As Integer
Public Bet As Long

Enum Strat
  Stay = 1 'Punta sempre lo stesso numero
  Random = 2 'Ogni volta cambia numero
  Last = 3 'Punta l'ultimo numero uscito
End Enum

Type Wh_Game 'Una variabile personalizzata che include tutte le caratteristiche della simulazione
  Wh_Numbers As Integer
  St As Strat
  Min_Bet As Integer
  Min_Win As Long
  Max_Loss As Long
  Number As Integer
End Type
```

```
'Subroutine che mostra a video il risultato di una singola simulazione: Vincita e Puntata Totale
Public Sub One_Game()
Dim G As Wh_Game
   G.Wh_Numbers = 36
   G.St = Last
   G.Min_Bet = 10
   G.Min_Win = 1000
   G.Max_Loss = 1000000
   G.Number = 0

   Call P_Wins(G, True)
   Debug.Print "WIN: " & Net_Win
   Debug.Print "T_BET: " & Tot_Bet
End Sub

'Subroutine che esegue un ciclo simulativo; dalla prima puntata fino alla vincita finale
Public Sub P_Wins(G As Wh_Game, Optional Pr As Boolean = False)
Dim W_Num As Integer
Dim S As String

   Win = 0
   Net_Win = 0
   Tot_Loss = 0
   Tot_Bet = 0
   N_Bet = 0
   Bet = G.Min_Bet

   Do Until (Net_Win >= G.Min_Win) Or (Tot_Loss >= G.Max_Loss)
      W_Num = Rand_From(0, G.Wh_Numbers)
      Tot_Bet = Tot_Bet + Bet
      N_Bet = N_Bet + 1
      If W_Num = G.Number Then
         Call Update_Record(G)
      Else
         Net_Win = 0
         Tot_Loss = Tot_Loss + Bet
      End If

      S = "W_N: " & W_Num & " B_N: " & G.Number & " Bet: " & Bet & " T_Bet: " & Tot_Bet & " N_W:
" & Net_Win
```

```
      If Pr Then Debug.Print S

      Bet = Choose_Bet(G, Tot_Loss)
      Select Case G.St
         Case Random: G.Number = Rand_From(0, G.Wh_Numbers)
         Case Last: G.Number = W_Num
      End Select

   Loop

End Sub

'Subroutine che aggiorna la vincita e/o la perdita, il totale puntato ecc.
Private Sub Update_Record(G As Wh_Game)
   Win = Bet * (G.Wh_Numbers - 2) ' Prize net of last bet
   If Win > Tot_Loss Then
      Net_Win = Net_Win + (Win - Tot_Loss)
      Tot_Loss = 0
   Else
      Tot_Loss = Tot_Loss - Win
      Net_Win = 0
   End If
End Sub

'Funzione che stabilisce l'ammontare della puntata successiva
Private Function Choose_Bet(G As Wh_Game, L As Double) As Long
   Choose_Bet = Round((G.Min_Win + L) / (G.Wh_Numbers - 2)) + 1
   If Choose_Bet < G.Min_Bet Then Bet = G.Min_Bet
End Function

'Funzione che genera un numero random interno entro un intervallo definito dall'utente
Private Function Rand_From(Low As Integer, High As Integer) As Integer
   Dim Rg As Integer
   Dim Rn As Double
      Rg = High - Low
      Rn = Rnd() * (Rg + 1) + Low
      Rand_From = Int(Rn)
End Function
```

Per ottenere dati medi (di vincita e di puntata) è necessario ripetere la procedura di cui sopra un numero sufficiente di volte. Questo viene fatto dalla seguente Macro[2] scritta non in un modulo, ma nello spazio di lavoro del foglio Excel nel quale si vuole utilizzare tale macro.

```
Public Sub Simulation()
Dim G As Wh_Game
Dim AvN_Bets As Double, Av_Win As Double, Av_Bet As Double
Dim i As Integer, Runs As Integer
   G.Wh_Numbers = Range("C4") 'Le celle in cui leggere i valori da utilizzare nella simulazione
   G.Min_Bet = Range("C5")
   G.Min_Win = Range("C6")
   G.Max_Loss = Range("C7")
   G.Number = Range("C8")
   Runs = Range("C10")

   For i = 1 To 3
      AvN_Bets = 0
      Av_Win = 0
      Av_Bet = 0
      G.St = i
      For j = 1 To Runs
         Call P_Wins(G)
         AvN_Bets = AvN_Bets + N_Bet
         Av_Win = Av_Win + Net_Win
         Av_Bet = Av_Bet + Tot_Bet
      Next j
      Cells(5, 5 + i).Value = Av_Win / Runs
      Cells(6, 5 + i).Value = AvN_Bets / Runs
      Cells(7, 5 + i).Value = Av_Bet / Runs
   Next i
End Sub
```

[2] Si parla di Macro quando si scrive, nello spazio di lavoro di un foglio Excel, una Subroutine pubblica senza parametri di input.

Il risultato viene quindi scritto nelle celle predisposte nel foglio di lavoro utilizzato, come mostrato nella figura seguente:

B	C	D	E	F	G	H
DATI ROULETTE						
Numeri	**36**			**Numero Fisso**	**Ultimo Vincente**	**Numero Casuale**
Puntata minima	**10**		**Vincita Media**	333.5068	316.7216	336.6122
Vincita minima prima di lasciare	**1000**		**Numero Puntate**	12.7602	12.8958	12.732
Perdita Massima	**500**		**Puntata Media Totale**	446.8504	452.1334	445.8082
Numero di partenza	**0**					
Numero di test	**5000**					

Il risultato del programma di simulazione del gioco delle roulette

5.9. Un secondo esempio di programma – Il gioco delle tre carte

Vediamo un secondo esempio. Il croupier prende tre carte, due perdenti ed una vincente, e le dispone a suo piacimento sul tavolo. Il giocatore che, a differenza del croupier, non sa dove sia la carta vincente ne sceglie una. A questo punto il croupier scopre una carta perdente, tra le due non scelte dal giocatore. Sul tavolo restano due carte coperte, quella scelta dal giocatore e quella che non è stata scoperta dal croupier. Il croupier offre al giocatore la possibilità di cambiare scelta. Conviene cambiare carta o la scelta è indifferente? La risposta non è difficile per chi conosce un po' di calcolo delle probabilità, ma può essere ottenuta in maniera sperimentale simulando tale gioco un numero sufficientemente grande di volte, ed osservando il numero di vincite che si hanno cambiando e non cambiando la carta. Questa simulazione viene eseguita dal seguente programma (che in effetti è generalizzato ad un numero qualsiasi di carte), e la cui comprensione viene lasciata come esercizio al lettore.

```
Enum Strategy
    Change = 1
    Stay = 2
    Random = 3
End Enum


Type Game
    NDoors As Integer
    St As Strategy
    NRuns As Long
End Type
```

```
Public Sub Main()
Dim My_Game As Game

   My_Game.NDoors = 3
   My_Game.NRuns = 10000
   My_Game.St = Change

   Debug.Print P_Wins(My_Game)

End Sub

'La funzione che effettua la simulazione e restituisce il numero di vittorie – Si parla di porte invece che di carte
Private Function P_Wins(G As Game) As Double
Dim i As Long, j As Integer
Dim Tot_Win As Long
Dim Sel_Door1 As Integer, Win_Door As Integer, Opn_Door As Integer, Sel_Door2 As Integer
Dim S As String
Dim Oth_Doors() As Integer

   For i = 1 To G.NRuns
      ReDim Oth_Doors(1 To G.NDoors)
      For j = 1 To G.NDoors
         Oth_Doors(j) = j 'Oth_Doors è un vettore che codifica le porte disponibili {1,2,3,…}
      Next j
      Win_Door = Rand_From(1, 3) 'La porta che vince
      Call Remove(Win_Door, Oth_Doors) 'Si rimuove la porta vincente dalle porte disponibili

      Sel_Door1 = Rand_From(1, 3) 'La porta scelta dal giocatore
      Call Remove(Sel_Door1, Oth_Doors) 'Si rimuove la porta selezionata da quelle disponibili

      Opn_Door = Oth_Doors(Rand_From(1, UBound(Oth_Doors))) 'La porta che viene aperta
      Call Remove(Opn_Door, Oth_Doors) 'Si rimuove la porta aperta da quelle disponibili

      'La porta vincente, non era selezionabile per essere aperta,
       'ma è selezionabile dal giocatore. Viene riaggiunta
```

```
      If Sel_Door1 <> Win_Door Then
         If CheckArray(Oth_Doors) Then 'Il vettore delle porte selezionabili potrebbe essere vuoto. Quando?
            ReDim Preserve Oth_Doors(1 To UBound(Oth_Doors) + 1)
         Else
            ReDim Oth_Doors(1 To 1)
         End If
         Oth_Doors(UBound(Oth_Doors)) = Win_Door
      End If

      Sel_Door2 = Change_Dr(G.St, Oth_Doors, Sel_Door1)

      S = "W_Door: " & Win_Door & " S_Door: " & Sel_Door1 & " O_Door: " & Opn_Door _
          _ & " _LSel_Door: " & Sel_Door2
      Debug.Print S

      If Win_Door = Sel_Door2 Then Tot_Win = Tot_Win + 1

   Next i

   P_Wins = Tot_Win / G.NRuns

End Function

Private Function Change_Dr(St As Strategy, V() As Integer, Or_Door) As Integer
Dim Ch As Boolean
   Change_Dr = Or_Door
   If St = Stay Then Ch = False
   If St = Change Then Ch = True
   If St = Random Then
      If Rnd() > 0.5 Then
         Ch = True
      Else
         Ch = False
      End If
```

```
    End If
    If Ch Then Change_Dr = V(Rand_From(LBound(V), UBound(V)))
End Function

Private Function Find_Pos(Value As Integer, V() As Integer) As Integer
Dim Found As Boolean
Dim Pos As Integer
   Find_Pos = -1
    Pos = LBound(V)
    Do While Pos <= UBound(V)
       If V(Pos) = Value Then
          Find_Pos = Pos
          Exit Do
       Else
          Pos = Pos + 1
       End If
    Loop
End Function

Private Sub Remove(Value As Integer, V() As Integer)
Dim Pos As Integer
    Pos = Find_Pos(Value, V)
    If Pos < 0 Then Exit Sub
    Do While Pos < UBound(V)
       V(Pos) = V(Pos + 1)
       Pos = Pos + 1
    Loop
    If LBound(V) = UBound(V) Then
       Erase V
    Else
       ReDim Preserve V(1 To (UBound(V) - 1))
    End If
End Sub
```

```
Private Function Rand_From(Low As Integer, High As Integer) As Integer
    Dim Rg As Integer
    Dim Rn As Double
        Rg = High - Low
        Rn = Rnd() * (Rg + 1) + Low
        Rand_From = Int(Rn)
End Function


'Il vettore potrebbe essere vuoto. Tale funzione verifica questa evenienza sfruttando la gestione degli errori
Private Function CheckArray(V() As Integer) As Boolean
On Error Resume Next
    CheckArray = True
    L = UBound(V)
    If Err.Number = 9 Then CheckArray = False
End Function
```