Estructuras algorítmicas en Visual Basic

www.aprendalia.org

1.- Introducción

Un algoritmo es un conjunto ordenado y finito de operaciones que permite hallar una solución de un problema.

2.- Características de los algoritmos

La calidad de un algoritmo, no depende solo de la efectividad de su funcionamiento, sino de la claridad de su código, es muy importante que sea fácilmente comprensible y que esté bien estructurado, debido a que esta característica facilitará su mantenimiento, actualización y adaptación a nuevas situaciones.

Un algoritmo bien hecho deberá cumplir como mínimo las siguientes condiciones:

- Ser fiables(los resultados habrán de ser exactos y precisos).
- Ser eficientes(utilizará de forma óptima los recursos del ordenador).
- Ser robusto(tener prevista una respuesta clara sean cuales sean los datos de entrada introducidos).
- Ser transportable(deberá estar diseñado de forma que se pueda poner en funcionamiento en cualquier ordenador).
- Ofrecer todas las facilidades posibles al usuario(con una interfaz amigable, mensajes claros y concisos, y una buena documentación).

3.- Estructura de un algoritmo

3.1 Datos

Los datos que se pueden manejar en un programa se pueden clasificar de la manera siguiente:

3.1.1 Datos de tipo numérico

Son números con los cuales se van a realizar operaciones aritméticas, podemos tener los siguientes:

- Enteros: Son los números enteros.
- Reales: Son los números decimales.

3.1.1.1 Datos de tipo numérico en Visual Basic

Los tipos de datos numéricos pueden ser: Integer, Long (entero largo), Single (signo flotante de simple precisión), Double (signo flotante de doble precisión) y Currency (moneda).
Ejemplo:

- *Dim entero as integer*
- *Dim enterolargo as long*
- *Dim simple as single*
- *Dim doble as double*
- *Dim moneda as currency*

3.1.2 Datos de tipo cadena

Son para almacenar cadena de caracteres y nunca un valor numérico.

3.1.2.1 Datos de tipo cadena en Visual Basic

Para declarar estos tipos de datos se debe declarar una variable string. El tamaño máximo de caracteres que puede almacenar son 255 caracteres.
Ejemplo:

- *Dim cadena as string*

3.1.3 Datos de tipo carácter

Son todos los caracteres en general.

3.1.3.1 Datos de tipo carácter en Visual Basic

Para declarar estos tipos de datos se debe crear una cadena de tamaño uno.
Ejemplo:

- *Dim caracter as string*1*

En el ejemplo anterior se declara una variable cadena de tamaño uno.

3.1.4 Datos de tipo lógico

Son aquellos que lo componen sólo los valores verdadero y falso.

3.1.4.1 Datos de tipo lógico en Visual Basic

Para declarar variables de este tipo se crea una variable de tipo Boolean.
Ejemplo:

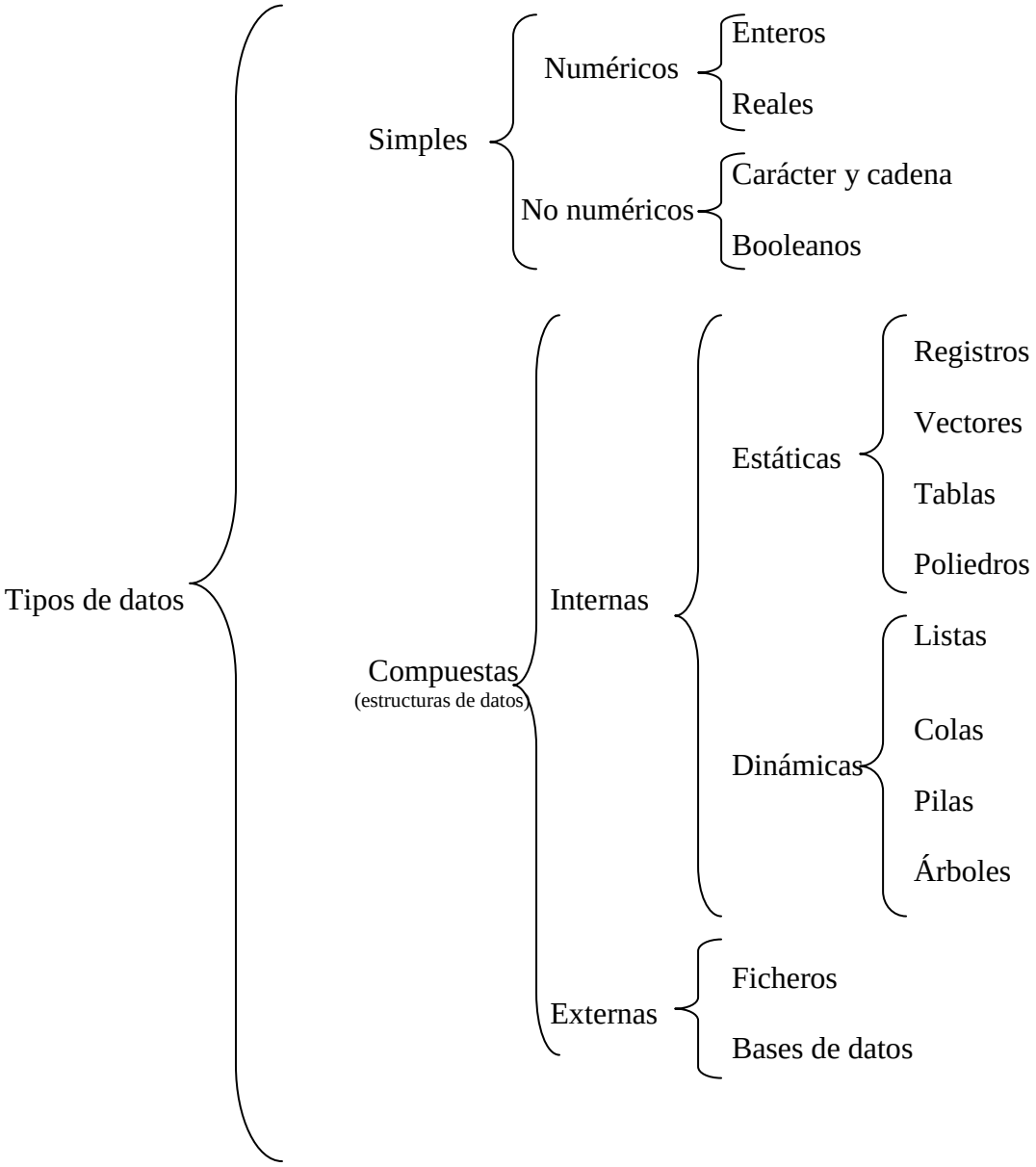
- *Dim logico as Boolean*

3.1.5 Estructuras de datos

Tenemos que pueden ser de dos tipos, internas y externas. Veamos ahora cada una de ellas a modo general, sin entrar en profundidad en cada una de ellas.

- Internas:
  - Estáticas: Son aquellas que utilizan la misma posición de memoría en el disco duro, y pueden ser:
    - Registros
    - Vectores
    - Tablas
    - Poliedros
  - Dinámicas: Son aquellas que utilizan distinta posición de memoria cuando son almacenadas, y pueden ser:
    - Listas
    - Colas
    - Pilas
    - Árboles
- Externas: Tenemos que pueden ser de dos tipos diferentes:
  - Ficheros
  - Bases de datos

Como resumen se puede descomponer todo lo visto en el siguiente esquema:



3.2 Expresiones

Una expresión normalmente viene representada por la unión de varios operandos que son: variables, constantes y/o literales, unidos entre sí por operandos que realizan una acción sobre ellos ya sea de relación, comparación o aritméticas. Veremos ahora cada una de las distintas expresiones:

3.2.1 Operandos

Son los datos que intervienen en las expresiones y podrán aparecer de forma explícita o podrán hacerse referencia a ellos a través de variables. Los operandos que podemos tener son:

- Constantes: Son elementos que no varían durante la ejecución del programa.
- Variables: Son elementos cuyo contenido puede modificarse durante la ejecución de un programa.

3.2.2 Operadores

Son elementos que unen los distintos operandos de una expresión realizando entre ellos operaciones ya sean de tipo lógico, aritmético, de relación, etc. Se pueden clasificar en:

- Aritméticos: Son +, -, *, /, MOD(resto división entera), DIV(división entera), ^(potencia).
- Alfanuméricos: +(concatenación)
- Relaciónales: =, <, >, <=, >=, <>
- Lógicos: NOT, AND, OR
- Orden de evaluación de los operandos: Es la misma que en matemáticas (paréntesis, signo, potencia, producto y división, suma y resta).

3.3 Instrucciones

Dentro de un programa podemos tener cientos, miles o millones de instrucciones, pero todas ellas se pueden clasificar en los siguientes tipos que veremos a continuación.

3.3.1 Simples

Dentro de este tipo de instrucciones podemos encontrar las siguientes:

3.3.1.1 Declarativas

Son instrucciones que se utilizan para declarar los objetos que se van a usar en el programa.

3.3.1.2 Primitivas

Son instrucciones que se ejecutan de forma inmediata por el procesador. Tenemos tres tipos:

- Entrada: Se utilizan para tomar datos del exterior y guardarlos en variables.
- Salida: Se utilizan para presentar, en pantalla o impresora, comentarios, mensajes al operador, contenido de variables, valor de constantes, resultados de una expresión, etc.
- Asignación: Este tipo de instrucciones se utilizan para asignar valores a las variables dentro del programa.

3.3.1.3 De Control

Se denominan instrucciones de control a las instrucciones que permiten controlar la ejecución de otras instrucciones bajo ciertas circunstancias. Son probablemente las más importantes de todo programa de ordenador. Podemos tener:

- Alternativas: Permiten realizar acciones alternativas, de forma que en el programa se ejecutarán una serie de instrucciones u otras, dependiendo de si cumple o no una determinada condición. Podemos tener:
  - Simples: Se utiliza cuando en un programa se quiere ejecutar un bloque de instrucciones sólo bajo una determinada condición.
  - Dobles: Se usa cuando se quiere ejecutar un bloque de instrucciones bajo alguna condición concreta, pero también se quiere ejecutar otro bloqeu independiente en el caso de que esta condición no se cumpla.
  - Múltiples: Se utilizan en casos donde la ejecución de un grupo de instrucciones u otro dependen de valores concretos y conocidos de una variable.
- Repetitivas: Se denomina bucle o ciclo a un proceso que dentro de un programa se repite un cierto número de veces. Podemos tener:
  - Mientras: Esta instrucción para cuando queremos que se repita una serie de acciones mientras se cumpla la condición puesta en el bucle.
  - Repetir....hasta: Esta instrucción se utiliza si queremos que se ejecute al menos una vez el bucle y luego en función de si se cumple la condición se ejecuta el bucle o no.
  - Para: Esta instrucción se utiliza cuando sabemos el número de veces que se va a ejecutar un bucle.
  - Iterar: Esta instrucción tiene la ventaja de poder evaluar la condición en cualquier lugar del bucle y salir en caso de cumplirse.

3.3.1.4 Comentarios

Son líneas en las cuales el programador introduce comentarios que favorecen la comprensión del programa, la localización rápida de partes concretas de código y la corrección de posibles errores. Es muy importante que un programa este bien comentado ya que para el mantenimiento del software nos ayudará muchísimo. Los comentarios deben ser los justos, sin poner comentarios en todas las líneas, solamente en aquellas instrucciones o conjuntos de instrucciones complicadas.

3.3.2 Compuestas

Las instrucciones compuestas están formadas por un conjunto de instrucciones simples que desarrolladas en un lugar apartado del resto y agrupadas bajo un nombre, son llamadas desde cualquier punto del programa. Una vez ejecutadas el control del programa vuelve a la instrucción siguiente a la que realizó dicha llamada.

4.- Pseudocódigo

Se expresará todo el formato de instrucciones mediante pseudocódigo. Las instrucciones utilizarán distintos símbolos que indicarán todas las posibilidades de dicha instrucción; por ejemplo, se utilizarán llaves para indicar que en la construcción de la instrucción hay que elegir obligatoriamente una de las opciones que encierran dichas llaves y los corchetes indicarán que lo que va dentro es opcional, las palabras en mayúscula serán palabras que aparecerán en la instrucción de la misma forma que aparecen en el formato, sin embargo, las palabras en minúscula deberán ser sustituidas en la instrucción por lo que se indique.

4.1 Primitivas

Las instrucciones primitivas se describen de la siguiente forma en pseudocódigo:

4.1.1 Salida

Esta instrucción muestra por pantalla el literal, el contenido de la variable, el valor de la constante o el resultado de la expresión aritmética que lleve como operando. Su formato es el siguiente:

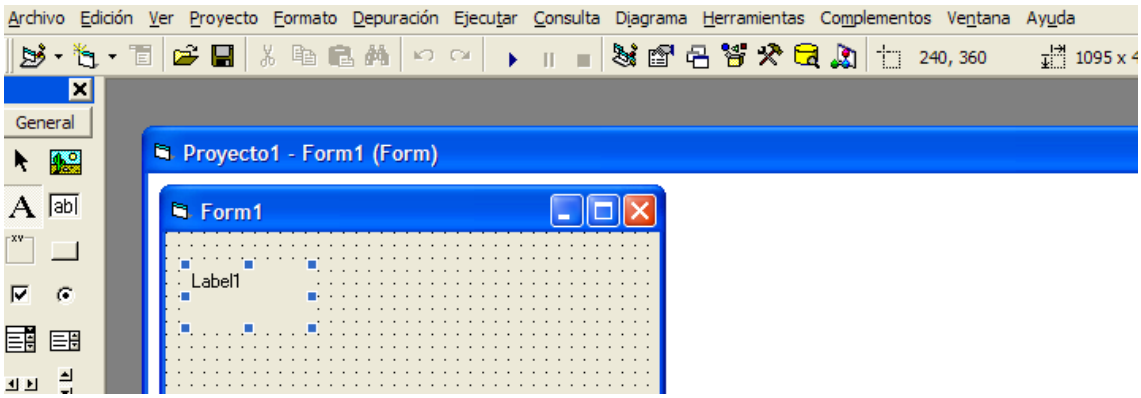
ESCRIBIR { *"literal"* | *nombre-variable* | *Nombre-constante* | *Expresión aritmética* }

EJEMPLOS:

ESCRIBIR "Introduzca un valor"
ESCRIBIR numero1
ESCRIBIR pi
ESCRIBIR numero1+numero2

4.1.1.1 Salida en VISUAL BASIC

Para que aparezca un texto, simplemente debemos seleccionar la opción LABEL en la parte de la barra de herramientas, (la que se encuentra a la izquierda de la pantalla), y la arrastramos como y colocamos sobre el formulario como se puede ver en la figura. Una vez puesta sobre él, podemos modificarlo y poner el texto deseado.



4.1.2 Entrada

Esta instrucción permite la introducción por teclado de un valor que se almacenará en la variable que lleve como operando. Su formato es el siguiente:

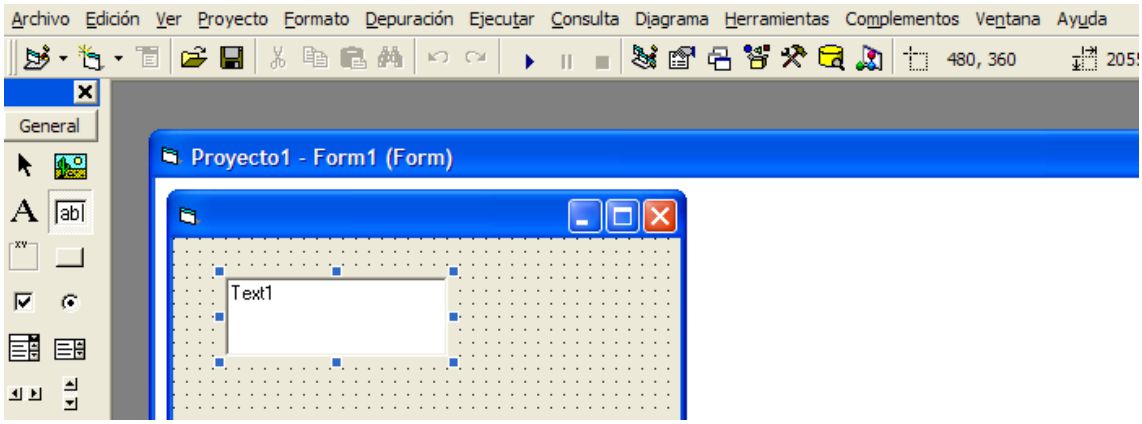
LEER *nombre-varible*

EJEMPLO:

LEER numero1

4.1.1.2 Entrada en VISUAL BASIC

Para leer valores, lo que de debemos hacer es seleccionar la opción TEXT BOX en la parte de la barra de herramientas, (la que se encuentra a la izquierda de la pantalla), y la arrastramos como y colocamos sobre el formulario como se puede ver en la figura. Una vez puesta sobre él, podemos modificar el tamaño y posición.



4.1.3 Asignación

Utilizando cualquiera de los símbolos expresados en el formato de la instrucción se asignará un valor a la variable destino, este valor podrá ser el contenido de otra variable, el valor de una constante, el resultado de una expresión aritmética, o un literal. Su formato es el siguiente:

Nombre-variable-destino= { *Nombre-variable* | *Nombre-constante* | Expresión aritmética | Literal }

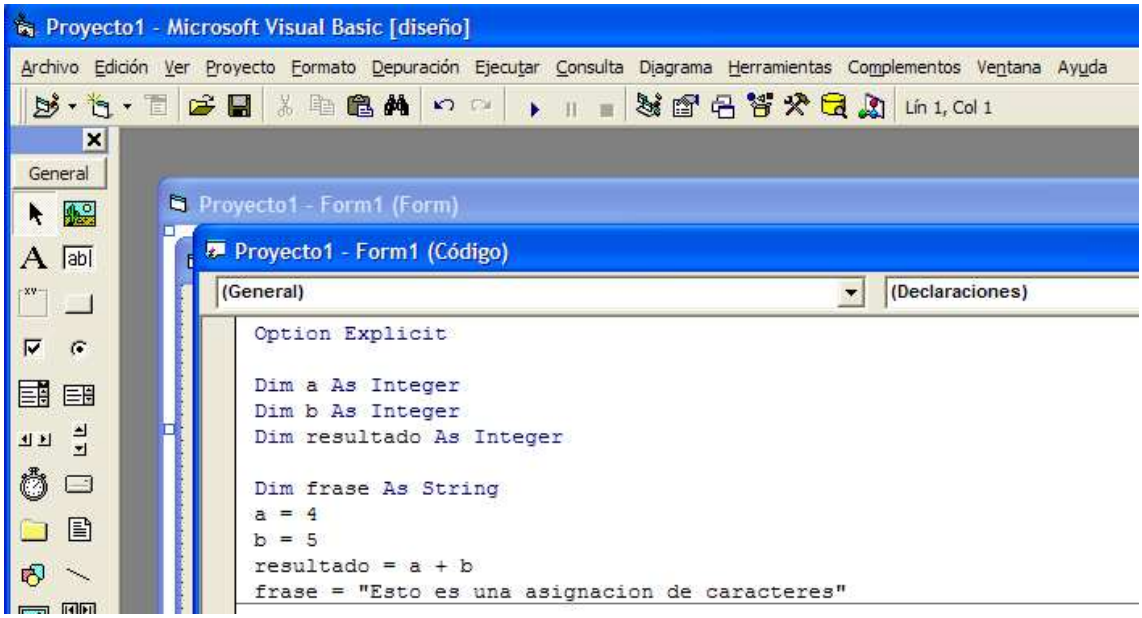
EJEMPLOS:

Numero1 = valor1
Numero2 = pi
Resultado = numero1+numero2
Nombre = “colegio”

4.1.3 Asignación en VISUAL BASIC

Para asignar valores en VISUAL BASIC lo que debemos hacer es poner en la parte izquierda del igual la variable en la cual queremos almacenar el valor, seguido de un = y luego en la parte derecha la variable o valor que queremos volcar su contenido sobre ella.

Ejemplos:



4.2 Sentencias de control

Existen las siguientes instrucciones en pseudocódigo capaces de controlar la secuencia de ejecución de otras instrucciones.

4.2.1 Alternativas

Como ya se indicó antes, estas instrucciones permiten la ejecución de un bloque de instrucciones en el caso de que se cumpla una determinada condición. Las condiciones que se han de evaluar será una expresión del siguiente tipo:

Literal Variable Constante Exp-aritmética	< > = <= >= <>	Literal Variable Constante exp-aritmética

4.2.1.1 Simple

Esta instrucción provoca una evaluación de la condición, en el caso de que esta se cumpla se pasaría a ejecutar el grupo de instrucciones que están detrás del ENTONCES y a continuación las que estuvieran después del FIN-SI. Pero si la condición no se cumple este conjunto de instrucciones no se ejecutaría y el control del programa pasará a las instrucciones que estuvieran debajo del FIN-SI. Su formato es el siguiente:

SI condición **ENTONCES**
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
FIN SI

EJEMPLO:

SI numero>0 **ENTONCES**
    ESCRIBIR “El valor es positivo”

FIN SI

Las instrucciones que van dentro de una instrucción SI pueden ser también instrucciones condicionales, con ello obtendremos instrucciones SI anidadas.

4.2.1.1.1 Alternativa Simple en VISUAL BASIC

```
If numero > 0 Then
   frase = "El número es positivo"
End If
```

4.2.1.2 Doble

En este otro caso, después de evaluar la condición, si esta se cumple, se ejecutarán el grupo de instrucciones del ENTONCES y si no se cumple se ejecutan el grupo de instrucciones del SINO, en cualquiera de estos dos casos después se continúa la ejecución de la instrucción que estuviera debajo del FIN-SI. Su formato es el siguiente:

SI condición **ENTONCES**
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
SINO
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
FIN SI

EJEMPLO:

SI numero>0 **ENTONCES**
    ESCRIBIR “El valor es positivo”
SINO
    ESCRIBIR “El valor es negativo”
FIN SI

4.2.1.2.1 Alternativa Doble en VISUAL BASIC

```
If numero > 0 Then
   frase = "El número es positivo"
Else
   frase = "El número es negativo"
End If
```

4.2.1.2.2 Alternativa Triple en VISUAL BASIC

4.1.2.3 Múltiple

Existe una instrucción condicional múltiple que simplifica algunos casos de la instrucción SI. Esta instrucción se basa en el conocimiento de antemano de lso valores concretos que tomará la variable que se quiere evaluar, indicando para cada uno de estos valores la secuencia de instrucciones a realizar; en el caso de que la variable no contenga ninguno de estos valores expresados, se ejecutarían el grupo de instrucciones incluidas en el EN OTRO CASO, en cualquiera de las circunstancias que se den y una vez ejecutadas el bloque de instrucciones correspondientes, el control del programa pasará a la instrucción que estuviera debajo del FIN. Su formato es el siguiente:

EN CASO DE (variable)
Valor 1: Instrucción 1
Instrucción 2
...
Valor 2: Instrucción 1
Instrucción 2
...
...
Valor N: Instrucción 1
Instrucción 2
...
EN OTRO CASO: Instrucción 1
Instrucción 2
...

EJEMPLO:

EN CASO DE operación
"+": resultado = numero1 + numero2
"-": resultado = numero1 - numero2
"":* resultado = numero1 * numero2
"/": resultado = numero1 / numero2
EN OTRO CASO:
ESCRIBIR "El valor de la operación no es válido"

4.1.2.3.1 Alternativa Multiple en VISUAL BASIC

```
Select Case operacion
   Case "+"
          resultado = a + b
   Case "-"
        resultado = a - b
   Case Else
        resultado = 0
End Select
```

4.2.2 Repetitivas

Dependiendo de si se conoce e antemano el número de repeticiones, o si al depender este número de una condición, esta se evalúa antes, durante o después de ejecutar el ciclo por primera vez, existen varias instrucciones repetitivas.

De la misma forma que las instrucciones condicionales pueden ir una dentro de otra, las instrucciones repetitivas también pueden anidarse.

4.2.2.1 Mientras

Antes de entrar en el ciclo se evalúa la condición del bucle, en el caso de que no se cumpla, el control del programa pasa a la instrucción que estuviera debajo del FIN-MIENTRAS sin haber ejecutado el bucle ni una sola vez.

En el caso de que la condición se cumpliera se entraría a ejecutar las instrucciones del ciclo que se repetirán hasta que esta dejase de cumplirse. Su formato es el siguiente:

MIENTRAS condición **HACER**
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
FIN MIENTRAS

EJEMPLO:

numero = 0
MIENTRAS numero<10 **HACER**
    numero = numero + 1
FIN MIENTRAS
ESCRIBIR “El valor de la variable número es: “
ESCRIBIR numero

4.2.2.1.1 Mientras en Visual Basic

```
numero = 3
While numero < 10
  numero = numero + 1
Wend
```

4.2.2.2. Repetir ... hasta

Una vez ejecutadas las instrucciones del ciclo por primera vez se evaluaría la condición de forma que, si la condición cumple, el control del programa pasará a la instrucción que estuviera debajo del HASTA, en caso contrario, se ejecutaría el bucle hasta que esta se cumpliera. Su formato es el siguiente:

REPETIR
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
HASTAS QUE condición

EJEMPLO:

```
numero = 5
REPETIR
    numero = numero - 1
HASTA QUE numero <0
ESCRIBIR “El valor de la variable número es: “
ESCRIBIR numero
```

4.2.2.2. Repetir ... hasta en Visual Basic

```
numero = 3
Do
    numero = numero + 1
Loop Until numero > 10
```

4.2.2.3 Para

Este bucle necesita una variable numérica que se inicialará automáticamente al valor inicial que se indique en la instrucción y que se incrementará o decrementará en tantas unidades como se indique cada vez que se ejecuten las instrucciones del bucle, la condición se evalúa después de este incremento o decremento y el bucle se termina cuando la variable supera el valor final. Por defecto, el incremento es de +1. Su formato es el siguiente:

PARA variable **DESDE** valor-inicial **HASTA** valor-final [INCREMENTO incremento] **HACER**
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
FIN PARA

EJEMPLO:

```
Numero = 10
PARA i DESDE 1 HASTA 10 HACER
    numero = numero + i
FIN PARA
ESCRIBIR “El valor de la variable número es: “
ESCRIBIR numero
```

4.2.2.3.1 Para en VISUAL BASIC

```
numero = 3
For i = 1 To 10
   numero = numero + 1
Next
```

4.2.2.4 ITERAR

En esta instrucción se ejecuta el primer grupo de instrucciones la primera vez de forma incondicional, y se evalúa la condición al llegar a SALIR SI; en ese momento si la condición se cumple se ejecutaría la instrucción que estuviera debajo del FIN-ITERAR, pero si no se cumple se siguen ejecutando las instrucciones del bucle que se cumpla la condición de salida. Su formato es el siguiente:

ITERAR
    Instrucción 1
    Instrucción 2
    ...
    Instrucción N
SALIR SI condición
    Instrucción 1
    Instrucción 2
    ...
    Instrucción M
FIN ITERAR

EJEMPLO:

numero = 6
ITERAR
    numero = numero - 1
    numero = numero - 2
SALIR SI numero<0
ESCRIBIR “El valor de la variable número es: “
ESCRIBIR numero

4.2.2.4 ITERAR en Visual Basic

numero = 3
Do
    numero = numero + 1
Loop Until numero > 10

4.3 Acciones compuestas

Formadas por un grupo de instrucciones simple que cumplirán una misión muy concreta y determinada, las instrucciones compuestas deberán ser definidas en un lugar aparte del resto de las instrucciones del programa, fuera de la secuencia, y llamadas desde el lugar del programa que se necesiten.

4.3.1 Declaración

La definición o declaración de la acción compuesta se realiza de la siguiente forma:

SUBPROGRAMA nombre-subprograma(variables)
Instrucción 1
Instrucción 2
...
Instrucción N
FIN SUBPROGRAMA

EJEMPLO:

SUBPROGRAMA raiz (numero:entero, resultado:entero)
resultado = sqrt(numero)
FIN SUBPROGRAMA

4.3.1.1 Declaración en Visual Basic

Private Sub cuadraddo
Resultado=numero*numero
End Sub

4.3.2 Llamada

Para llamar a una instrucción compuesta dentro del código del programa bastará con utilizar su nombre. Esto provocará que el control del programa se desplace al módulo llamado, se ejecuten las instrucciones que lo componen y se devuelva posteriormente el control a la instrucción siguiente a la llamada. Su formato es el siguiente:

nombre-subprograma(variables)

EJEMPLO:

Raiz(numero, respuesta)

4.3.1.2 Llamada en Visual Basic

Para realizar la llamada solamente debemos poner el nombre de la función y el formulario en el cual se encuentra.

4.4. Comentarios

En cualquier parte del programa se podrán incluir líneas de comentario, para ello basta con comenzar la línea mediante //. Su formato es el siguiente:

//comentario

4.4.1 Comentarios en Visual Basic

Para colocar comentarios en Visual Basic, simplemente debemos poner el símbolo ‘ delante de la línea a comentar. Es el símbolo que aparece en el cierra interrogante del teclado.

Ejemplo:

```
'numero = 3
```

5.-Como resolver un algoritmo

Para resolver un algoritmo debemos seguir los siguientes pasos:

1- Leer varias veces el enunciado del algoritmo para entender lo que nos pide el problema.
2- Analizar cuantas variables vamos a necesitar y el tipo de las variables. Necesitaremos algunas variables para leer los datos que nos suministra el usuario y otras variables para operar con los datos.
3- Pensar como se puede resolver el problema de la manera más sencilla posible. Durante menos complicaciones y menos instrucciones tenga nuestro algoritmo más rápido se realizará en el ordenador.
4- Una vez realizado el algoritmo, realizar una traza para comprobar si esta correcto. Una traza es una prueba en la cual introducimos unos valores de entrada y comprobamos si la salida es correcta.

6. Resumen de estructuras

Estructura	Ejemplo

Alternativa Simple	If numero > 0 Then
	  frase = "El número es positivo"
	End If

Alternativa Doble	If numero > 0 Then
	  frase = "El número es positivo"
	Else
	  frase = "El número es negativo"
	End If

Alternativa Triple	If numero > 5 Then
	  frase = "El número es mayor que cinco"
	Else
	  If numero > 0 Then
	  frase = "El número es mayor que cero y menor que cinco"
	  Else
	  frase = "El número es negativo"
	  End If
	End If

Alternativa Múltiple	Select Case operacion
	  Case "+"
	  resultado = a + b
	  Case "-"
	  resultado = a - b
	  Case Else
	  resultado = 0
	End Select

Repetitiva Mientras	numero = 3 While numero < 10 numero = numero + 1 Wend
Repetitiva Repetir...hasta	numero = 3 Do numero = numero + 1 Loop Until numero > 10
Repetitiva Para	numero = 3 For i = 1 To 10 numero = numero + 1 Next
Repetitiva Iterar	numero = 3 Do numero = numero + 1 Loop Until numero > 10

7. Resumen de variables

Valor	Descripción
Byte	Valor byte
Integer	Valor entero
Long	Valor entero largo
Single	Valor de coma flotante de precisión simple
Double	Valor de coma flotante de precisión doble
Currency	Valor de moneda
Decimal	Valor decimal
Date	Valor de fecha u hora
String	Valor de cadena de caracteres
Boolean	Valor de tipo Boolean; **True** o **False**

8. Resumen de operadores

Aritméticos	Comparación	Lógicos
Exponenciación (^)	Igualdad (=)	**Not**
Negación (-)'	Desigualdad (<>)	**And**
Multiplicación y división (*****,/)	Menor que (<)	**Or**
División de enteros (\)	Mayor que (>)	**Xor**
Módulo aritmético (**Mod**)	Menor o igual que (<=)	**Eqv**
Adición y substracción (+,-)	Mayor o igual que (>=)	**Imp**
Concatenación de cadenas (**&**)	**Is**	&

9. Resumen del lenguaje Visual Basic

Categoría	Palabras clave
Control de matrices	Array Dim, Private, Public, ReDim IsArray Erase LBound, UBound
Asignaciones	Set
Comentarios	Comentarios mediante ' o Rem
Constantes y literales	Empty Nothing Null True, False

Categoría	Palabras clave
Control de flujo	Do...Loop For...Next For Each...Next If...Then...Else Select Case While...Wend
Conversiones	Abs Asc, AscB, AscW Chr, ChrB, ChrW CBool, CByte CCur, CDate CDbl, CInt CLng, CSng, CStr DateSerial, DateValue Hex, Oct Fix, Int Sgn TimeSerial, TimeValue
Fechas y horas	Date, Time DateAdd, DateDiff, DatePart DateSerial, DateValue Day, Month, MonthName Weekday, WeekdayName, Year Hour, Minute, Second Now TimeSerial, TimeValue
Declaraciones	Const Dim, Private, Public, ReDim Function, Sub
Formato de cadenas	FormatCurrency FormatDateTime FormatNumber FormatPercent
Control de errores	On Error Err
Entrada y salida	InputBox LoadPicture MsgBox
Literales	Empty False Nothing Null True
Matemática	Atn, Cos, Sin, Tan Exp, Log, Sqr Randomize, Rnd
Varios	Función RGB
Objetos	CreateObject Objeto Err GetObject

Categoría	Palabras clave
Operadores	Adición (+), Sustracción (-) Exponenciación (^) Módulo aritmético (Mod) Multiplicación (*), División (/), División entera (\) Negación (-) Concatenación de cadenas (&) Igualdad (=), Desigualdad (<>) Menor que (<), Menor que o igual a (<=) Mayor que (>), Mayor que o igual a (>=) Is And, Or, Xor Eqv, Imp
Opciones	Option Explicit
Procedimientos	Call Function, Sub
Redondeo	Abs Int, Fix, Round Sgn
Id. de motor de secuencias de comandos	ScriptEngine ScriptEngineBuildVersion ScriptEngineMajorVersion ScriptEngineMinorVersion
Cadenas	Asc, AscB, AscW Chr, ChrB, ChrW Filter, InStr, InStrB InStrRev Join Len, LenB LCase, UCase Left, LeftB Mid, MidB Right, RightB Replace Space Split StrComp String StrReverse LTrim, RTrim, Trim
Variant	IsArray IsDate IsEmpty IsNull IsNumeric IsObject TypeName VarType