Asumiendo que la expresión booleana sea verdadera la primera vez que se evalúa, se ejecuta el estatuto y, al terminar, se evalúa nuevamente la condición booleana. Si continúa verdadera, el estatuto se ejecuta de nuevo; si se encuentra falsa, el lazo termina, transfiriendo el control del programa al estatuto siguiente del lazo. Por ejemplo:

```
var
    pH : real;
begin
    LlenaTanque;              { llenar el tanque de agua }
    Muestrear (pH);           { tomar la medida de pH inicial }
    While pH < 7.2 do
    begin
        PonerAlcalino;        { mezclar sustancia en el tanque }
        AgitarTanque;         { agitar }
        Muestrear (pH);       { tomar lectura del sensor }
    end;
end.
```

Esta parte del código es de un sistema de control de un aparato de proceso químico. Todo lo que hace es llenar un tanque de agua y asegurarse de que el pH sea cuando menos de 7.2. Si el agua del tanque se vuelve muy ácida, el pH sube de valor y el agua no sería potable.

Primero se llena el tanque. Después se toma el valor inicial de pH. El agua puede ser potable desde el principio, en cuyo caso el lazo no se ejecutaría jamás. Pero si el agua es ácida, el lazo se ejecuta, añadiendo una cantidad de sustancia alcalina y tomando la muestra de pH otra vez. Si el pH sobrepasa el valor de 7.2, el lazo termina; de lo contrario, el lazo se ejecuta otra vez, añadiendo alcalinidad al agua y muestreando el pH una vez más.

La más importante propiedad del lazo WHILE/DO es que la expresión booleana se revisa ANTES de que el estatuto se ejecute, al contrario de lo que sucede con el lazo REPEAT/UNTIL.

LAZOS REPEAT/UNTIL

El lazo REPEAT/UNTIL es similar al lazo WHILE/DO. Como el otro, REPEAT/UNTIL ejecuta un estatuto hasta que la expresión booleana sea verdadera. La forma general es:

```
REPEAT estatuto UNTIL expresión booleana;
```

Lo más importante es que el estatuto se ejecuta cuando menos una vez. A diferencia del lazo WHILE/DO, no es necesario inicializar las variables. Es correcto inicializar las variables necesarias dentro del estatuto.

Otra cosa interesante es que las palabras reservadas REPEAT/UNTIL actúan como separadores de bloque, igual que lo harían las palabras BEGIN/END, de tal manera que el estatuto puede ser simple o compuesto, sin cambiar NADA de la sintaxis dada anteriormente.

Estos dos tipos de lazos son bastante similares, y es válido cuestionar cuál de los dos es el necesario en un momento dado y para una situación en particular. WHILE/DO puede hacer todo lo que hace REPEAT/UNTIL, sin embargo, en un pricipio puede ser confuso el uso de ellos. Recomendación: Use REPEAT/UNTIL en todos los casos en que el estatuto deba ejecutarse cuando menos una vez. Cuando codifique lazos, siempre pregúntese qué pasaría si el lazo no se ejecutara jamás; si no hay una respuesta coherente, codifique un REPEAT/UNTIL en vez de un WHILE/DO.

CAPITULO 4.- PROCEDIMIENTOS Y FUNCIONES

Mucha gente piensa que los lazos como WHILE/DO y REPEAT/UNTIL (y la innecesaria posición del GOTO) son la piedra angular de una programación estructurada, pero no hay tal. En el fondo, la programación estructurada se compone de detalles más sutiles. La capacidad humana de englobar un sistema complejo se ve disminuida rápidamente, a menos que se encuentre una estructura o patrón. Cuando se hace un programa en FORTRAN, por ejemplo, de más de 1000 líneas (lo cual puede tomar unas seis semanas de trabajo), y se intenta recordar cómo funciona exactamente la parte inicial de dicho programa, resalta la diferencia de la estructura que un lenguaje como Pascal puede ofrecer.

Uno de los detalles escondidos de la programación estructurada es la secuencia de código que ejecuta tareas discretas. Tales secuencias de código son llamadas "subprogramas".

En Pascal hay dos tipos de subprogramas: procedimientos y funciones. Estos subprogramas son secuencias de estatutos colocados fuera del cuerpo principal del programa. Ambos, para ser ejecutados, se invocan llamándolos simplemente por su nombre. La única diferencia entre ellos es que la función regresa un valor de un tipo declarado, mientras que los procedimientos no.

Los procedimientos y funciones son, en efecto, programas en miniatura. Pueden tener declaraciones de etiquetas, constantes, tipos, variables, procedimientos y funciones. Por supuesto que tienen estatutos de código. La única diferencia entre un procedimiento y un programa es la palabra PROGRAM y la puntuación final después del END.

```
PROCEDURE Mensaje (Linea : String80);
const
        Xpos = 50;
        Ypos = 24;
begin
        gotoxy (Xpos, Ypos);
        write (Linea);
end;
```

Las funciones son un poco diferentes. Una función tiene un tipo y toma un valor que regresa a la lógica del programa que la llamó:

```
FUNCTION Area (R : real) : real;
const
        Pi = 3.14159;
begin
        Area := Pi * R * R;
end;
```

El tipo de la función Area es real. Como se puede ver en la única línea de código de la función, se evalúa una expresión con el valor del radio R y el valor asignado al nombre de la función. Fuera de esas dos distinciones, los procedimientos y las funciones son iguales.

```
program Piramide;
uses
   Crt;

procedure HacerPiramide (Ch : char);
var
   i,j : char;
begin
   writeln ('Piramide de la ',Ch);
   for i:= 'A' to Ch do
   begin
      gotoxy (105-ord(i),wherey);
      for j:='A' to pred(i) do write (j);
      for j:=i downto 'A' do write (j);
      writeln;
   end;
   gotoxy (40,25);
   write ('Presione <Enter> para continuar...');
   readln;
end;

var
   Ch : char;
begin
   clrscr;
   repeat
      writeln ('** PIRAMIDE DE LETRAS **');
      writeln;
      writeln ('Utilice * para terminar...');
      writeln; writeln;
      Ch := upcase(readkey);
      if Ch in ['A'..'Z'] then HacerPiramide (Ch);
      clrscr;
   until Ch = '*';
end.
```



```
program ValidaNumeros;
uses
   Crt;


function Digito (Numero : word; Posicion : byte) : integer;
var
   NumStr : string;
begin
   str (Numero,NumStr);
   if length (NumStr)  Posicion then Digito := -1
   else Digito := ord(NumStr[Posicion])-48;
end;

var
   Numero   : word;
   Posicion  : byte;
   R         : integer;


begin
   clrscr;
   repeat
      writeln;
      write ('Dame el numero  : '); readln (Numero);
      write ('Dame la posicion: '); readln (Posicion);
      R := Digito(Numero,Posicion);
      if R=-1 then writeln ('**ERROR**')
      else writeln ('El digito #',Posicion,' de el numero es ',R);
   until R=-1
end.
```

PARAMETROS FORMALES Y ACTUALES

Pasar datos a los procedimientos y a las funciones puede hacerse de dos formas:

1) usando variables globales para que cualquier procedimiento y/o función pueda leerse.

2) a través de una lista de parámetros para cada procedimiento y/o función.

La primera es una muy mala idea, por no mencionar las limitantes que el hardware pueda imponer. Es una buena práctica que un procedimiento tenga solamente que manejar una lista de valores.

Los parámetros definidos en la declaración del procedimiento se llaman "parámetros formales". Los valores que están presentes en una lista de parámetros de la invocación en particular se llaman "parámetros actuales". Si los parámetros son pasados por referencia, los parámetros actuales deberán tener variables de tipos iguales para guardar en ellos los parámetros formales.

TRANSFERENCIA DE DATOS POR VALOR Y REFERENCIA

PARAMETROS PASADOS POR VALOR

Un parámetro pasado por valor es justamente eso: Un valor copiado del parámetro actual al parámetro formal. El movimiento del valor hacia el procedimiento es en un sentido. Nada regresa ni puede ser usado por el programa que lo llamó.

Hay ventajas poderosas en el movimiento de valores dentro del procedimiento. El procedimiento puede guardar, modificar y mutilar el parámetro en la medida que las necesidades del procedimiento lo exijan, y no existen efectos colaterales fuera del procedimiento. La copia del parámetro actual es una copia privada, estrictamente local para el mismo procedimiento.

La forma de declarar pase de valores por valor es:
PROCEDURE (variable : tipo);

PARAMETROS PASADOS POR REFERENCIA

En muchas ocasiones el punto central de pasar un parámetro a un procedimiento o función es tener ese mismo, pero modificado para su uso. Para que un procedimiento pueda modificar un parámetro y regresarlo así, éste debe pasarse por referencia.

A diferencia de los parámetros pasados por valor, los valores pasados por referencia NO pueden ser literales, constantes o expresiones. Los valores de constantes y literales, por definición, no pueden ser cambiados.

Para pasar un valor por referencia, un parámetro actual debe ser una variable del mismo tipo que el parámetro formal. No hay tipos compatibles; deben ser exactamente del mismo tipo. La única excepción a esta regla son los tipos string, los cuales pueden ser definidos por una longitud de hasta 255.

La forma para definir valores pasados por referencia es:
PROCEDURE (VAR variable : tipo);

CAPITULO 5 ARREGLOS (Arrays)

Las variables que se han venido manejando hasta ahora son todas tipos simples. Iniciaremos ahora el estudio de algunos tipos estructurados que posee Pascal. Se puede decir que un tipo estructurado difiere de un tipo simple en que el tipo estructurado posee mas de un componente. Y cada componente del tipo estructurado es una variable, la cual a su vez puede ser de tipo estructurado o simple. Lo importante con este tipo de variables es la forma en como se tiene acceso a cada uno de sus componentes.

ARREGLO DE UNA DIMENSION (VECTORES)

Un arreglo no es mas que una colección de variables, en donde todas son del mismo tipo. Se puede decir que una línea de texto es un arreglo de caracteres; un vector se puede representar como un arreglo de números; una matriz se puede pensar como un arreglo, cuyos elementos son a su vez vectores.

Un arreglo se declara en términos de un índice y un tipo base, o sea, el tipo de cada uno de sus componentes.
Visto de otra forma, podemos decir que un arreglo nos va a permitir:

- Denotar un grupo de variables mediante un solo identificador.
- Poder distinguir un elemento en particular del grupo mediante un índice. De esta forma podemos referenciar el iésimo componente del arreglo.

El índice podrá ser cualquier tipo ordinal. Así, por ejemplo, una declaración del tipo array podrá quedar de la siguiente forma:

```
Numeros : array[1..100] of real;
```

Esta declaración permite disponer de un arreglo de 100 elementos. En este caso el índice es del tipo entero y el tipo de cada elemento es real.

Según se mencionó antes, el índice no es necesariamente numérico; puede ser cualquier tipo ordinal:

```
Grupo : array['A'..'G'] of integer;
```

Un elemento individual de un arreglo variable se denotará como una variable indexada, escribiendo el nombre (identificador) del arreglo seguido por el correspondiente valor del índice, enmarcado entre corchetes cuadrados. Ejemplo:

```
Grupo['E']
```

está haciendo referencia a aquél elemento del arreglo Grupo cuyo subíndice es 'E', el cual tiene o puede tener almacenado un valor del tipo integer.

Con esta variable (Grupo['E']) es posible hacer todas las operaciones en las cuales el operando sea de tipo entero:

```
Acumulado := Grupo['A'] + Grupo['E'];

Prom      := Grupo['B'] + Grupo['C'] / 2;
```