```
for i:=1 to 24 do { EJE Y }
begin
   gotoxy (1,i);
   write ('|')
end;

for i:=1 to 80 do { EJE X }
begin
   gotoxy (i,Eje);
   write ('-')
end;

for i:=1 to 80 do
begin
   radian := round(Eje-Escala*cos(Angulo));
   gotoxy (i,radian);
   write ('*');
   Angulo := Angulo + 0.1
end
end.
```

DECISIONES

ESTATUTO IF/THEN/ELSE

Un estatuto condicional redirecciona el programa en una de dos direcciones, en base a un valor booleano. En Pascal el estatuto condicional es IF/THEN/ELSE. La claúsula ELSE es opcional, pero las palabras IF y THEN siempre van asociadas. En su forma más simple, cada estatuto se construye así:

IF expresión booleana THEN estatuto1 ELSE estatuto2

La forma de evaluación es la misma de como lo sugiere la frase en inglés (de la misma forma que BASIC): Si la condición booleana es verdadera, se ejecuta el estatuto #1; si es falsa y el estatuto #2 existe, éste se ejecuta.

Cualquiera de los dos (o ambos) estatutos asociados con IF/THEN/ELSE pueden ser estatutos compuestos. Recuerde que un estatuto compuesto puede ser usado en cualquier lugar como un estatuto simple. Por ejemplo:

```
IF I < 0 THEN
begin
      Negativo := true;
      I := abs(I)
end
      ELSE Negativo := false;
```

Nota importante: No hay punto y coma después del estatuto compuesto BEGIN/END. El fragmento completo es considerado un solo estatuto IF. NUNCA debe haber un punto y coma antes de la palabra reservada ELSE.

Ya que IF es un estatuto perfectamente válido, es posible que uno o varios estatutos contengan IF/THEN/ELSE. Los IF's pueden ser "anidados" tanto como se quiera, pero eso hace el código menos legible. La forma estructurada es:

```
IF [expresión booleana1]THEN
      IF [expresión booleana2]THEN
            IF [expresión booleana3]THEN
                  estatuto;
```

que puede ser transformado con el operador AND de la siguiente forma:

IF expresión booleana1 AND expresión booleana2 AND expresió booleana3 THEN estatuto

La discusión anterior no incluye cláusulas ELSE. Anidar IF's no excluye ELSE, pero cambia totalmente el significado de los estatutos:

```
IF expresión booleana1 THEN estatuto1
ELSE
      IF expresión booleana2 THEN estatuto2
      ELSE
            IF expresión booleana3 THEN estatuto3
```

La mejor forma de tratar este caso es con el estatuto CASE OF, que se verá enseguida. El estatuto CASE OF es una forma compacta de representar IF's anidados.

ESTATUTO CASE/OF

Escoger entre algunas alternativas de caminos de control, resulta crítico en un programa de computación. Se ha visto cómo IF /THEN/ELSE, en su forma más simple, sirve para escoger entre dos alternativas, basado en una expresión booleana. El problema de legibilidad aparece cuando se anidan más de dos o tres. Considere el problema de enviar un mensaje a la pantalla, en base a un número clave de entrada. El reporte de problemas de un sistema computarizado puede ser algo así:

```
Beep;
writeln ('****PRECAUCION****');
IF CodProblema = 1 THEN
      writeln ('[001] Combustible menor del 10%')
ELSE IF CodProblema = 2 THEN
      writeln ('[002] La presión de aceite abajo del mínimo')
ELSE IF CodProblema = 3 THEN
      writeln ('[003] Temperatura de operación muy alta')
ELSE IF CodProblema = 4 THEN
      writeln ('[004] Voltaje de baterías bajo')
ELSE IF CodProblema = 5 THEN
      writeln ('[005] Líquido de frenos menor al especificado')
ELSE IF CodProblema = 6 THEN
      writeln ('[006] Fluido de transmisión faltante')
ELSE IF CodProblema = 7 THEN
      writeln ('[007] Radiador con bajo nivel de agua')
ELSE
      writeln ('[***] Falla de lógica en el reportador')
```

Esto trabaja adecuadamente, pero el código es un poco caótico al final. Esta selección se puede codificar en un solo estatuto CASE OF creado para seleccionar en base a una sola evaluación. Reescribiendo resulta:

```
Beep;
writeln ('****PRECAUCION****');
CASE CodProblema OF
     1: writeln ('[001] Combustible menor del 10%');
     2: writeln ('[002] La presion de aceite abajo del minimo');
     3: writeln ('[003] Temperatura de operacion muy alta');
     4: writeln ('[004] Voltaje de baterias bajo');
     5: writeln ('[005] Liquido de frenos menor al especificado');
     6: writeln ('[006] Fluido de transmision faltante');
     7: writeln ('[007] Radiador con bajo nivel de agua')
ELSE
     writeln ('[***] Falla de logica en el reportador')
end; { final de CASE }
```

Aquí la variable CodProblema se llama "el selector del CASE". Puede ser una variable o una expresión. El valor se mantiene durante la selección de alternativas del estatuto. Los números en las líneas del CASE se llaman "etiquetas del CASE". Cada etiqueta va seguida de dos puntos (:) y después de un estatuto. El estatuto puede ser simple o compuesto.

Cuando el CASE se ejecuta, el selector es evaluado y comparado, uno por uno, contra las etiquetas del mismo. Si una etiqueta es igual al valor del selector del CASE, el estatuto asociado con la etiqueta se ejecuta. Una vez que se ejecutó un estatuto, el CASE OF transfiere el control al resto del programa. Solo uno (o ninguno) de los estatutos del CASE OF se ejecuta cada vez que se evalúa un estatuto CASE OF.

Si ninguna de las etiquetas concuerda con el valor del selector, se ejecuta el estatuto del ELSE (si existe). ELSE es opcional; si no hay el control se pasa al resto del programa.

La forma general del CASE OF es:

```
CASE OF
     ante1 : estatuto1;
     ante2 : estatuto2;
     ante3 : estatuto3;
     .....
     anteN : estatutoN;
ELSE estatuto
END;
```

Se pueden tener tantas etiquetas como se quiera, hasta 256. En el ejemplo, cada etiqueta es una constante numérica. Una etiqueta NO puede ser una variable. Se puede componer de una lista de constantes separadas por comas.

El siguiente ejemplo es de un sistema de análisis por correo. Las respuestas son agrupadas en distribuciones geográficas de los estados de la república. Estado es un tipo enumerado de los estados en orden alfabético. Este orden toma el número de respuestas para cada región en particular:

```
TYPE
     Estado =   (AGS,  BCN, BCS,CAM, COH, COL, CHP, CHI, DF,
                 DGO,  GTO,  GRO,  HGO,  JAL, MEX, MCH, MOR,
                 NAY,  NL, OAX, PUE, QRO, QR, SLP, SIN, SON,
                 TAB, TAM, TLX, VER, YUC, ZAC)
VAR
  Norte, California,
  Centro Sur, Golfo, Peninsula          :byte;
  DelEstado                             :Estado;
BEGIN
     CASE DelEstado OF
          SON, CHI COH,
          DGO, NL, TAM                  :Norte     := Norte + 1;
          BCN, BCS                      :California := California + 1;
          ZAC, SLP, AGS,
          GTO, QRO, HGO,
          TLX, DF, PUE,
          MOR, MEX, VER                 :Centro    := Centro + 1;
          COL, MCH, GRO,
          OAX, CHP                      :Sur       := Sur + 1;
          VER, TAB                      :Golfo     := Golfo + 1;
          CAM, YUC, QR                  :Peninsula := Peninsula + 1;
     END;
END.
```

Lo más importante que debemos recordar en lo que se refiere a la lista de etiquetas, es que éstas deben ser constantes. Un valor en particular debe aparecer solamente una vez en dicha lista.

```
program Horno;
uses
  Crt;
const
  Abierta       = true;
  Cerrada       = false;
  ZonaMuerta    = 4;
  PuntoOp       = 12;
  LimSup        = PuntoOp+ZonaMuerta;
  LimInf        = PuntoOp-ZonaMuerta;
  InerciaTermica = 1;
var
  tiempo,
  Temperatura : integer;
  Valvula     : boolean;
```

```
procedure Ejes;
var
   Cont : byte;
begin
   gotoxy (1,24);
   for Cont := 1 to 80 do write ('-');
   for Cont := 1 to 24 do
   begin
      gotoxy (1,Cont);
      write ('|');
   end; { for }
   for Cont := 1 to 80 do
   begin
      gotoxy (Cont,LimSup);
      write ('-');
      gotoxy (Cont,PuntoOp);
      write ('-');
      gotoxy (Cont,LimInf);
      write ('-');
   end;
end; {-------------------------------------------------------}


begin
   clrscr;
   Ejes;
   Temperatura := 0;
   Valvula := Cerrada;
   for tiempo: = 1 to 80 do
   begin
      if Temperatura  LimSup then Valvula := Cerrada;
      if Temperatura  LimInf then Valvula := Abierta;


      gotoxy (tiempo, 24-Temperatura);
      write ('*');
      sound (2000);
      delay (10);
      nosound;
      delay (200);
      case Valvula of
         Cerrada : Temperatura := Temperatura - InerciaTermica;
         Abierta : Temperatura := Temperatura + InerciaTermica;
      end; { case }
   end; { for }
end.
```


CICLOS

LAZOS FOR

En ocasiones es necesario ejecutar un estatuto varias veces con un rango de valores. El ejemplo más usado es un programa que genere una lista de las raíces cudradas de los números comprendidos entre uno y cien. Pascal provee de un estatuto que ejecuta el mismo código para cada valor, y continúa con el resto del programa cuando todos los casos han sido evaluados. Se llama lazo FOR, y es uno de los tres lazos disponibles en Pascal. Su funcionamiento es similar al FOR de BASIC.

La forma general del estatuto es la siguiente:

```
FOR variable de control := valor inicial TO valor final DO
    estatuto
```

Los valores iniciales y finales pueden ser expresiones. La variable de control es de cualquier tipo ordinal, incluyendo tipos enumerados.

Es necesario aclarar que el estatuto FOR no especifica que la variable de control "se incremente en uno". La variable de control se asigna al SIGUIENTE valor en su estado ordinal. Esto es muy importante cuando la variable de control es de tipo enumerado.

Imprimir las raíces es bastante fácil:

```
FOR I := 1 TO 100 DO
begin
    J := sqrt (I);
    writeln ('La raíz cuadrada de ',I,' es: ',J)
end;
```

El estatuto FOR, en su forma pura, incrementa la variable de control; esto es, asigna el SIGUIENTE valor "hacia arriba" del estado ordinal. Algunas veces es necesario hacerlo al revés: Que a la variable de control se le asigne el valor precedente "hacia abajo" (decremento) en el estado ordinal:

```
FOR I := 17 DOWNTO 7 DO estatuto;
FOR Color := Indigo DOWNTO Naranja DO estatuto;
FOR Ch := 'Z' DOWNTO 'X' DO estatuto;
```

LAZOS WHILE/DO

Como se ha visto, el lazo FOR ejecuta un estatuto un número determinado de veces; ni más ni menos. La variable de control no puede ser alterada durante la ejecución del lazo. Hay muchos casos en que el lazo se repite hasta que una condición dada ocurre. La variable de control debe ser cambiada durante la ejecución del lazo. Pascal ofrece dos maneras de construir este tipo de lazos: WHILE/DO y REPEAT/UNTIL. La forma general del lazo WHILE/DO es:

```
WHILE expresión booleana DO estatuto
```

Como en los lazos FOR, el estatuto puede ser simple o compuesto. Si el valor de la expresión es verdadero, entonces el estatuto es ejecutado.

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.