```
                                cos, arctan, y PI. }
var
   i : shortint;
begin
   writeln ('** FUNCIONES TRIGONOMETRICAS **':50); writeln;
   write  (' ':4, 'n');
   write  (' ':7, 'sin (n*PI) ');
   write  (' ':4, 'cos (n*PI) ');
   writeln (' ':2, 'arctan (n) ');
   for i := -8 TO 8 DO
   begin
      arg := PI * i / 8;
      write (i / 8: 7 : 4);
      write (sin (arg): 13 : 4);
      write (cos (arg): 13 : 4);
      writeln (arctan (i / 8): 13 : 4)
   end
end; {-------------------------------------------------------}


procedure IntDemo;      { El procedimiento IntDemo muestra las
                          funciones enteras: INT, ROUND, TRUNC y FRAC. }

var
   i : shortint;
begin
   writeln ('** FUNCIONES PARA ENTEROS **':60);
   write  (' n');
   write  (' ':6, 'INT (n)');
   write  (' ':3, 'ROUND (n)');
   write  (' ':2, 'TRUNC (n)');
   writeln (' ':3, 'FRAC (n)');
   for i:=-10 to 10 do
   begin
      arg := i/4;
      write  (arg:5:2);
      write  (Int(arg):10:1);
      write  (round(arg):10);
      write  (trunc(arg):10);
      writeln (frac(arg):13:2)
   end
end; {-------------------------------------------------------}


procedure AritDemo;   { El procedimiento AritDemo muestra algunas
                        funciones miscelaneas: ABS, SQR, y SQRT. }
var
   i : shortint;
begin
   writeln ('** FUNCIONES ARITMETICAS **':60);
   write  (' n');
```



```
   write  (' ':6, 'ABS (n)');
   write  (' ':3, 'SQR (n)');
   writeln (' ':2, 'SQRT (n)');
   for i:=-10 to 10 do
   begin
      write  (i:3);
      write  (abs(i):10);
      write  (sqr(i):10);
      if (i) then writeln ('- ':10) else writeln (sqrt(i):12:4)
   end
end; {-------------------------------------------------------}


procedure PotenciaDemo; { El procedimiento PotenciaDemo muestra
                          la forma de exponenciacion en Pascal }
var
   I      : shortint;
   Num : real;
begin
   writeln ('** EXPONENCIAL **':50);
   write  (' ':8, 'x');
   write  (' ':13, 'y');

   writeln (' ':13, 'x^Y');
   for I := -10 to 10 DO
   begin
      Num := I / 2;
      write (Num:10:4, Num:15:4);
      if Num0 then writeln (Exp(Num*ln(abs(Num))):15:4) else writeln ('-':13);
   end
end;{-------------------------------------------------------}


function EsPrimo (num : integer) : boolean;
var
   Primo  : boolean;
   Prueba : integer;
begin
   Primo := true;
   Prueba := 2;
   while Primo and (Prueba < = Imt (sqtr(num))) do
      if num mod prueba = 0 then primo: false else inc(prueba);
   EsPrimo := Primo;
end; { fin de funcion EsPrimo }


function Factorial (num : integer) : real;
begin
   if num = 0 then
      Factorial:=1
   else
```

```
      Factorial := num * Factorial(num-1);
end; { fin de funcion Factorial }


procedure PrimosFactorialDemo; { El procedimiento PrimosFactorialDemo muestra
                                 como calcular un factorial y determinar si
                                 es numero primo o no. }
var
  I : shortint;
begin
  writeln ('** FUNCIONES FACTORIAL Y PRIMOS **'); writeln;
  write   (' N');
  write   (' ':6, 'N es numero primo?');
  writeln (' ':6, 'Factorial (N)');
  for I:= 1 to 20 do
  begin
     write   (I:2);
     if EsPrimo(I) then write ('SI':15) else write ('No':15);
     writeln (Factorial(I):25:0);
  end
end;{----------------------------------------------------}


procedure HiperbolicaDemo; { El procedimiento HiperbolicaDemo muestra
                             como calcular funciones hiperbolicas sinh, cosh. }
var
  I : shortint;
begin
  writeln ('** FUNCIONES HIPERBOLICAS **':60);
  write   (' n':5);
  write   ('sinh (n)':17);
  writeln ('cosh (n)':16);
  for I:=-10 to 10 do
  begin
     arg:= I/4;
     write   (arg:6:3);
     write ((exp(arg)-exp(-arg))/2:15:4);
     writeln ((exp(arg)+exp(-arg))/2:15:4);
  end
end;{----------------------------------------------------}


  { El procedimiento Menu despliega un menu en la pantalla y responde
    apropiadamente a la seleccion del usuario. El parametro Salir
    regresa un valor TRUE cuando el usuario selecciona la opcion Terminar. }

procedure Menu (var Salir : boolean);
const
  col = 25;
var
  Seleccion : char;
```



```
begin
  Irscr;
  Salir := false;
  gotoxy (col,5);
  writeln ('Funciones Numericas');
  gotoxy (col,7);  writeln ('1. Exponencial');
  gotoxy (col,8);  writeln ('2. Trigonometrica');
  gotoxy (col,9);  writeln ('3. Enteros');
  gotoxy (col,10); writeln ('4. Aritmeticas');
  gotoxy (col,11); writeln ('5. Potencias');
  gotoxy (col,12); writeln ('6. Primos y factoriales');
  gotoxy (col,13); writeln ('7. Hiperbolicas');
  gotoxy (col,14); writeln ('8. Terminar');
  writeln; gotoxy (col-5,18);
  write ('  *** Seleccion de Menu: ');
  repeat
     Seleccion := readkey;
  until Seleccion in ['1'..'8'];
  clrscr;
  case Seleccion of
     '1' : ExpLnDemo;
     '2' : TrigDemo;
     '3' : IntDemo;
     '4' : AritDemo;
     '5' : PotenciaDemo;
     '6' : PrimosFactorialDemo;
     '7' : HiperbolicaDemo;
     '8' : Salir := true;
  end;
  if Seleccion  '8' then
  begin
     writeln;
     write ('Presiona la <barra Espaciadora> para continuar...');
     repeat
     until readkey = ' ';
  end;
  clrscr;
end; {----------------------------------------------------}


begin
  repeat
     Menu (Fin)
  until Fin
end.
```

SUBRANGOS Y TIPOS ENUMERADOS

Todos los tipos explicados anteriormente son tipos simples, predefinidos en TURBO Pascal y listos para ser usados. Gran parte de la fuerza de Pascal es su habilidad para crear estructuras de datos fuera de esos tipos simples. Las estructuras de datos pueden hacer los programas fáciles de escribir y, posteriormente, de leer.

Definir tipos para datos particulares es bastante fácil. La palabra reservada TYPE inicia la parte de la definición de tipos del programa, y es aquí precisamente donde se necesita algo de planeación para estructurar los datos:

```
TYPE
ElTipo = LaDefinicion;
```

De aquí en adelante los tipos que se discutan deben ser declarados y definidos en la parte de definición de tipos del programa.

Una definición de tipos no ocupa un lugar en el área de datos del programa, únicamente provee de instrucciones al compilador, para el manejo de datos que sean de ElTipo en el programa fuente.

SUBRANGOS

La estructura de datos más sencilla que se puede definir es un subconjunto de un tipo ordinal llamado "subrango". Si se escogen dos valores legales cualquiera, del tipo ordinal, esos dos valores, más los de enmedio, definen el subrango de tipo ordinal. Por ejemplo:

```
TYPE
    Mayusculas      ='A'..'Z';
    Minusculas      ='a'..'z';
    Digitos         ='0'..'9';
```

Mayúsculas son el rango de caracteres A,B,C,D,E hasta Z. Dígitos incluye los caracteres de 0,1,2,3 hasta 9. Las comillas son importantes; le dicen al compilador que los valores en el subrango son de tipo CHAR. Si la declaración se hace sin comillas para Dígitos:

```
Digitos = 0..9;
```

se tiene un subrango del tipo entero, pues '7' no es lo mismo que 7.

Una expresión en la forma 'A'..'Z' o 3..6 se llama "intervalo cerrado". Un intervalo cerrado es un rango de valores ordinales, incluyendo los dos que marcan los límites y todos los que caen en él.

TIPOS ENUMERADOS

Los novatos en Pascal frecuentemente encuentran difícil de digerir la notación de los tipos enumerados. El "TURBO Pascal Reference Manual" llama a los tipos enumerados "tipos escalares definidos por el usuario". Muchos textos de Pascal se refieren a ellos como tipos enumerados. Un tipo enumerado es un tipo ordinal que el programador define. Uno de los mejores ejemplos que el lenguaje ofrece es el tipo BOOLEAN.

El tipo BOOLEAN, de hecho, es un tipo enumerado predefinido por el compilador y usado de manera especial. El tipo BOOLEAN es una lista ordenada de dos valores con nombres únicos: False y True. No es un par de cadenas de caracteres conteniendo las palabras en Inglés "False" y "True". El tipo BOOLEAN es un número binario con los valores de 00 y 01. Se ha nombrado el código 00 dentro del tipo BOOLEAN como False y el código 01 dentro del tipo BOOLEAN como True.

Considere una lista de valores con nombres únicos: los colores del espectro luminoso. Para crear un tipo enumerado de él se declara:

```
TYPE
    Espectro = (Rojo, Naranja, Amarillo, Verde, Azul, Indigo, Violeta);
```

La lista de un tipo enumerado siempre se marca entre paréntesis. El orden en que se colocan los valores define el valor ordinal, el cual puede ser probado usando la función ORD(X). Por ejemplo, Ord(Amarillo) regresará el valor 2. Ord(Rojo) 0.

Se pueden comparar valores de un tipo enumerado con otros del mismo tipo. El estatuto Amarillo > Rojo (2 > 0) regresará el valor booleano True.

```
Verde = Violeta          o          Azul < Naranja
```

Regresarán ambos el valor booleano False.

Los valores del tipo Espectro son todos constantes. Pueden ser asignados a variables de tipo Espectro. Por ejemplo:

```
VAR
    Color1, Color2 : Espectro;
BEGIN
    Color1:=Amarillo;
    Color2:=Indigo;
```

Una gran desventaja de los tipos enumerados es que no pueden ser impresos en la pantala o en la impresora. No puede codificar Writeln (Naranja), y esperar ver la palabra "Naranja" en la pantalla. TURBO Pascal genera el error 66:I/O no permitido. En esos casos se puede crear un array de strings indexados por el tipo enumerado:

```
    Nombres         : array[Rojo..Violeta] of String[80];
    Nombres[Rojo]   :='Rojo';
    Writeln (Nombres[Rojo]);
```

ARRAYS

Un array (vector o arreglo) es una estructura de datos consistente en un número determinado de elementos idénticos, con un solo identificador como referencia para el grupo. El programa se refiere a cada elemento por medio de un número o índice. En Pascal, los índices no son necesariamente números. Tipos enumerados, caracteres y subrangos, pueden actuar algunas veces como índices, permitiendo una enorme riqueza de expresiones, imposible de obtener con algún otro lenguaje computacional.

Un elemento de un array puede ser de cualquier tipo, excepto de tipo FILE (archivo). Los arrays pueden constar de estructuras completas de datos. Es posible tener arrays de records (registros) o arrays de arrays. Un índice debe ser parte de algún subrango o grupo de tipo ordinal, o un tipo enumerado definido por el usuario.

Los siguientes son ejemplos para mostrar algunas declaraciones válidas:

```
CONST
      Distritos      =14;
TYPE
      String80       =string[80];
      Grados         ='A'..'F';
      Porcentaje     =1..99;
      Niveles        =(K,G1,G2,G3,G4,G5,G6,G7,G8,G9);
      Materias       =(Ingles, Matematicas, Historia, Arte, Espanol);
      Ficha          =record
                           Nombre     :String80;
                           Matrícula  :String80;
                           IQ         : integer;
                           Promedio   :Porcentaje;
                           Finales    :array[Materias] of Distritos;
                      end;
      DefGrados      =array[Grados] of String80;
VAR
      ArchivoK12          :array[Niveles] of Ficha;
      Aprobados           :array[Niveles] of boolean;
      SubTotal            :array[1..24] of integer;
      PorcArea            :array[1..Distritos] of Porcentaje;
      NivelArea           :array[1..Distritos] of array[Niveles] of Porcentaje;
```

Las declaraciones anteriores son parte de un programa en Pascal, para el control escolar de calificaciones. Cuando TURBO Pascal separa área en memoria para un arreglo, no inicializa con cero los elementos. Si hay basura en el RAM al tiempo de compilar, los valores de los elementos del arreglo tendrán valores indefinidos. Es responsabilidad del programador iniciar el arreglo con algún valor (aun con cero).


```
program MiPrimerArray;

uses
  Crt;
var
  Numero : array[1..100] of integer;
  N, i   : byte;
  Suma,
  Prom   : real;
begin
  writeln ('Cuantos numeros son? ');
  readln (N);
  Suma := 0;
  for i:=1 to N do
  begin
     write ('Numero : ');
     readln (Numero[i]);
     Suma := Suma + Numero[i];
  end;
  Prom := Suma / N;
  clrscr;
  writeln ('LOS NUMEROS FUERON :');
  for i:=1 to N do writeln (Numero[i]:10);
  writeln;
  writeln ('La suma es = ',Suma:10:3,' y el promedio es = ',Prom:10:3);
end.
```

RECORDS

Un registro es una estructura compuesta de muchos elementos de diferentes tipos, agrupados por cada elemento. Estos elementos son llamados "campos" de un registro. Un taller de reparación de vehículos podría necesitar un archivo conteniendo el inventario de sus partes de repuesto. Para cada parte de repuesto se necesita un registro con el número de parte, la descripción, el costo de venta, precio de compra, mínimo y existencia. Todos esos elementos están íntimamente encadenados en un grupo físico (la pieza del carro), así que para simplificar el proceso de programación, es necesario reunir todos los campos en un registro:

```
TYPE
    ParteReg = record
                   NumParte,Clase:integer;
                   Descripcion:string[80];
                   NuestroCosto:real;
                   PrecioLista:real;
                   Minimo:integer;
                   Existencia:integer;
               end;
VAR
    ParteActual, ParteProx    :ParteReg;
    ArchivoPartes             :file of ParteReg;
    Almacenados               :integer;
    DebeOrdenarse             :boolean;
```

La nueva entidad se convierte en un nuevo tipo con su propio nombre. Los datos del registro pueden ser asignados, escritos en archivos y otras operaciones como una sola unidad, sin necesidad de especificar ningún campo del registro.

```
ParteActual := ParteProx;                  { asigna una parte a otra }
read (ArchivoPartes, ParteProx);           { lee una parte del archivo }
```

Cuando se necesite trabajar con campos individuales dentro de un registro, la notación consistirá en el nombre del registro seguido de un punto y el identificador del campo.

```
Almacenados := ParteActual.Existencia;
IF Almacenados < ParteActual.Mínimo THEN DebeOrdenarse := True;
```

Los operadores relacionales NO pueden ser usados con los registros. Decir que un registro es "mayor que" o "menor que" otro, carece de sentido. En el ejemplo anterior se compararon dos campos de los registros, no los registros propiamente. Los campos eran numéricos, por lo que pueden ser comparados con el operador "<".

STRINGS

Manipular palabras y líneas de texto es una función fundamental en un programa de computadora. Como mínimo, el programa debe ser capaz de mostrar mensajes tales como "Presione una tecla para continuar...". En Pascal, como en la mayoría de los lenguajes, los caracteres de una línea pueden tomarse juntos y manejarse como una entidad llamada "string". Muchos implementadores de Pascal, incluyendo aquellos que diseñaron TURBO Pascal, han ampliado la definición en cuanto al tratamiento de strings, con strings de "longitud variable", los cuales son tratados por el compilador de manera especial. Los strings de longitud variable tienen una longitud "lógica", misma que varía dependiendo de lo que contenga el string. Strings de diferentes longitudes lógicas pueden ser asignados a otros, siempre y cuando las longitudes físicas no se excedan.

Una variable string se define usando la palabra reservada STRING. Una longitud física debe especificarse entre paréntesis cuadrados (con un máximo de 255). Ejemplo:

```
VAR
    Mensaje     :string[80];        { longitud física de 80 }
    Nombre      :string[30];        { longitud física de 30 }
    Dir         :string[30];        { longitud física de 30 }
    Estado      :string[2];         { longitud física de 2 }
BEGIN
    Mensaje     := 'Esta es una prueba...';
    Nombre      := 'F. Eugenio López G.';
    Dir         := 'Malinche #427 San Nicolás';
    Estado      := 'NL';
```